2009年冬号

あんどきゅめんてっどでびあん

東京エリアDebian勉強会
関西エリアDebian勉強会著

デビアン勉強会

目次

1 Introduction

上川 純一, 山下 尊也

1.1 東京エリア Debian 勉強会

Debian 勉強会へようこそ。これから Debian の世界にあしを踏み入れるという方も、すでにどっぷりとつかっているという方も、月に一回 Debian について語りませんか？

Debian 勉強会の目的は下記です。

- Debian Developer (開発者) の育成。
- 日本語での「開発に関する情報」を整理してまとめ、アップデートする。
- 場 の提供。
  - 普段ばらばらな場所にいる人々が face-to-face で出会える場を提供する。
  - Debian のためになることを語る場を提供する。
  - Debian について語る場を提供する。

Debian の勉強会ということで究極的には参加者全員が Debian Package をがりがりと作るスーパーハッカーになった姿を妄想しています。情報の共有・活用を通して Debian の今後の能動的な展開への土台として、「場」としての空間を提供するのが目的です。

1.2 関西 Debian 勉強会

関西 Debian 勉強会は Debian GNU/Linux のさまざまなトピック (新しいパッケージ、Debian 特有の機能の仕組、Debian 界隈で起こった出来事、などなど）について話し合う会です。

目的として次の三つを考えています。

- ML や掲示板ではなく、直接顔を合わせる事での情報交換の促進
- 定期的に集まれる場所
- 資料の作成

それでは、楽しい一時をお楽しみ下さい。

2 Debian Conference 2009 参加報告

岩松 信洋

2.1 Debconf とは

2009 年度の Debconf は 6 月 23 日から 6 月 30 日まで、スペインのエクストラマドゥーラで行われました。日本からは、武藤 健志、前田 耕平、山根 秀樹、岩松 信洋が参加しました。

2.1.1 Debconf の歴史・経緯

Debian Conference `http://debconf9.debconf.org/` は Debian の開発者たちが一同に介するイベントです。通常顔をあわせることのないメンバーたちが一同に介し友好を深め、技術的な議論を戦わせます。過去の開催履歴を見てみると表 1 のようになります。

表 1　歴代の Debconf 参加者推移

年	名前	場所	参加人数
2000	debconf 0	フランス ボルドー	
2001	debconf 1	フランス ボルドー	
2002	debconf 2	カナダ トロント	90 名
2003	debconf 3	ノルウェー オスロ	140 名
2004	debconf 4	ブラジル ポルトアレグレ	150 名
2005	debconf 5	フィンランド ヘルシンキ	200 名
2006	debconf 6	メキシコ オアスタペック	300 名
2007	debconf 7	英国スコットランド エジンバラ	約 400 名
2008	debconf 8	アルゼンチン マルデルプラタ	約 200 名
2009	debconf 9	スペイン エクストラマドゥーラ	約 250 名

2.1.2 Debconf 2009

2009 年度の Debconf の会場はエクストラマドゥーラ州のカセレス (Caceres) にある州が管理する施設を活用しました。スペインのネットワーク会社である telefonica の協力によって、専用のネットワーク回線を会場中にはりめぐらせ、無線 LAN メインのネットワークを構築しました。(アクセスポイントには OpenWrt を使っていたようです。) 宿泊は会場にある宿泊施設と、会場から徒歩 15 分程度に位置する Francisco de Sande に分散していました。

2.2 スペイン/カセレス

2.2.1 行き方

行き方としては空路を使った場合、日本からスペイン/マドリッド、マドリッドからカセレスという順になります。日本からマドリッド（バラハス国際空港）までは直行便がありません。どこかでトランジットが必要になります。各

メンバはタイ国際空港経由で入国しました。距離は約 10000km。飛行時間は約 21 時間かかります（日本からタイまで約 5 時間、タイからマドリッドまで約 16 時間)。バラハス国際空港からマドリッドの市街に移動し、さらに列車で 4 時間ほど移動する必要があります。列車のチケットは予約する必要があり、座る場所も決まっています。マドリッドにあるターミナル駅、アートチャ駅で購入することも可能ですが、日本チームはインターネットでチケットを予約し、印刷して持っていきました。

2.2.2 会場

会場は、エクストラマドゥーラの施設を借り切って開催されました。先に書いたとおり、宿泊施設が併設されており、4 割の参加者は併設した宿泊施設から参加していました。もう片方の宿泊施設である Francisco de Sande は世界遺産である遺跡の中にあり、ゲームに出てきそうな宿でした。併設された宿泊施設には発表者やメイン開発者が泊まっており、ここでも格差社会が見え隠れしています。debconf の前にやっている debcamp と呼ばれるものがあり、こちらは開発が主体になっています。開発メインで行いたい人は debcamp から参加するとよいでしょう。


- Upper Talk Room: メイン用。250 人ほど入ることができます。

- Lower Talk Room: サブ会場。50 人ほど入ることができます。
- BoF Room
  BOF 用。30 人ほど入ることができます。プレゼンテーション設備がないため、皆パソコンを開いてプレゼンテーション資料を見ていました。また、空調設備が貧弱なため、蒸し風呂状態でした。
- Hacklab 1/2 および 食堂: Hacklab はハック専用の部屋です。


2.3 スケジュール

23 日の Debian Day で Debian Conference は開幕し、30 日まで毎日いろいろな予定が組まれていました。27 日だけはカンファレンス参加者で day-trip を実施しました。

2.4 主となった討論

2.4.1 i18n 関係

今回も 4 回の i18n セッションが設けられ、リリースノート翻訳、DDTP、Wiki をつかった　翻訳などの検討が行われたようです。DDTP 翻訳パーティを検討されているようなので、日本からも参加できるようにしたいところです。

2.4.2 組み込み関係

組み込み関係のセッションは全部で 9 つあり、組み込みからマルチアーキテクチャサポートなどが議論されました。

2.4.3 Multiarch round table

2.4.4 Openmoko Buzz Fix Party

カンファレス参加者に格安で提供された OpenMoko のバグフィックスパーティ。OpenMoko はフリーな携帯電話なのですが、日本では使えないので、購入はしませんでした。買った人に見せてもらったのですが、なかなかよさげです。Android も動きますし。ただ、うまく携帯としてつかえないことがあるようで...（以下略。

2.4.5 Mer: Maemo Reconstructed

Maemo のコミュニティ向け環境 Mer についての説明。タブレット PC 向けのようですが、N8XX や OpenMoko でも動作するようです。

2.4.6 Scratchbox2 for crosscompiling debian

Riku がメンテナンスしている Scratchbox2 を組み込み環境でどのように利用するのか、という話。scratchbox でツールを作成し、scratchbox qemu のフロントエンドとして利用したパッケージ作成ができるようになったので皆で使いましょう。

2.4.7 uclibc and busybox

uclibc と busybox Debian でどのようにサポートしたらいいのかという話。emdebian で busybox ベースの debian 環境をサポートするために頑張っているようです。これらについての意見交換をしました。sh4 もサポートに入る予定です。

2.4.8 emdebian BoF

毎回恒例になっている emdebian の進捗報告。emdebian では、Emdebian Grip と Emdebian Crush の 2 つのプロジェクトが進行中。前者は busybox/uclibc ベースの debian を、後者は 現在の Debian をクロスコンパイルサポートするプロジェクト。emdebian-tools ができて、ある程度ましになったけど、まだ不具合が多いので開発者募集中。apt-cross は依存関係の実行がまだまだなので、今後力をいれて開発していくとのこと。私からは sh4 のサポートもするから協力しろ、と言っておきました。

2.4.9 QEMU for Debian Developers

aurel32 によるフリーの CPU エミュレータである qemu の進捗状況と使い方について。Debian でサポートしてるアーキテクチャのほとんどは qemu でサポートしているのでマシンが手元になくてもデバッグできるよ、という話。イメージも彼の Web サイトにあるので、mips や ppc などを試したい人はどうぞ。

2.4.10 Crossbuilding on Debian for a derived distro

既存の Linux 組み込みシステムから Debian のシステムに移行した時の話とツールの使い方、クロスコンパイルの方法についての話。発表者の Wooky は去年から会社が変わったのですが、システムを Debian に移行したので、そのときに行った方法を話していました。CMake でのクロスコンパイルが面倒だったようです。

2.4.11 xcontrol

Simon が毎年やっている xcontrol の話。debian/control ファイルを拡張してクロスコンパイルパッケージや、バックポートのサポートを容易にしましょうというセッション。前回から進展があったのかは不明です。

2.5 ツール関係

2.5.1 News on Debian Autobuilding

buildd の現状と問題点の報告。dep-wait、failed の遅延への対応や、パッケージプライオリティに対応したビルドについて議論されました。

2.5.2 Trust is good, control is better

パッケージのインストール、アンインストールのチェックができる piuparts の進捗報告。HP のマシン 2 台を使って、すべてのパッケージのチェックを定期的に行っています。これにより、次期リリースではパッケージのインストール、アンインストールが問題なくできるようになるでしょう。

2.5.3 Using FOSSology for license analysis in Debian

tbm によるコードライセンス解析ソフトの話。deb, rpm,tar ボールなどをリポジトリに食わせライセンスの構文解析を行った結果を DB にまとめているとのこと。ライセンスと Copuright がどのように変化していったのかまで調査できるようです。デモをしていましたが、ちょっと動きは遅いかな。今後に期待です。`http://fossology.org/`

2.5.4 not your grandpa's debhelper

Joey による debhelper の解説。debhelper は dh_のプログラムを提供しますが、これらをまとめたコマンド dh を作りましたという話。dh を使うと debian/rules を以下のようにまとめることができます。

```
#!/usr/bin/make -f
%:
  dh: $@
```

各ターゲットもオーバーライドすることができて、override_dh_xxxx を作成しておくと呼ばれるようになります。dh 勉強会とかやりたいですね。

2.5.5 UDD: Ultimate Debian Database

Debian/Ubuntu のパッケージ情報や popcon 情報、debtags、LDAP、アップロード履歴、DDTP、testing ミグレーション、バグ、NEW キュー、lintian、スクリーンショット URL 等々といった情報を全部 PostgreSQL データベースに突っ込んでデータマイニングしてみました、という話でした。これらのデータベースは merkel と alioth で提供されています。インターフェイスは公開されているので、誰でも試すことができます。

2.5.6 pam-auth-update: manhandling debconf for fun and profit

squeeze から新しく導入された、pam-auth の設定ツールです。今までは、/etc/pam.d/以下を手動で設定していましたが、/usr/share/pam-configs 以下のテンプレートを使い、postinst, prerm の中で、pam-auth-update コマンドを実行することで簡単に pam-auth の設定をできるようになります。3rd party 製のツールも、これに則れば簡単に導入出来るようになるので、ユーザにとってもとてもメリットが大きいでしょう。Sid の環境では、次のファイル

が既にこの制御下にあるようです。

- /etc/pam.d/common-account
- /etc/pam.d/common-auth
- /etc/pam.d/common-password
- /etc/pam.d/common-session

2.5.7 Changing the default system shell

デフォルトシェル /bin/sh をなぜ変えるのか？ポリシーもあるようですが、切り替える一番の動機としては、遅いからだということを言っていました。bash で記述されたものは約 800 あるうち、fix 済みのものは 160 だそうです。bash を検出するには、devscripts, lintian, archive rebuilds, piuparts, ユーザの報告があるそうです。最終的には自分で判断して bash にするのをやめ、dash を選択するようにしてほしいと言っていました。

2.5.8 DDE, Debian Data Export

Debian に関する情報を HTTP プロトコルで取得することができる RESTful アプリケーションのようです。`http://dde.debian.net/dde` や `http://debtags.debian.net/dde` から、データを取得することができます。

2.5.9 Libvirt: Hypervisor independent virtual machine management

libvirt という仮想マシンを扱うための API の話です。Debian 固有の話は特になく、libvirt の紹介をしていました。Debian でもちゃんと使えるようなので、ぜひ使ってみましょう。

2.5.10 RFH maintaining big packages

iceweasel のパッケージメンテナの Mike さんの BOF。iceweasel, OOo では、どうやってユーザや開発者をより多く巻き込んでいくか、活動を宣伝していくか、という点で議論が交わされました。

2.6 リリース関係の話

2.6.1 Keynote from the Release Team

Release team によるキーノート。Debian からのニュース `http://www.debian.org/News/2009/20090729` でも流れているようにこの場で、次期安定版である Squeeze のフリーズとタイムベースリリースの話が出ました。うまく行くか不安ではありましたが、その場ではいけるんじゃね？という空気になっていました。（案の定、ML ではフレームになっていますが。）

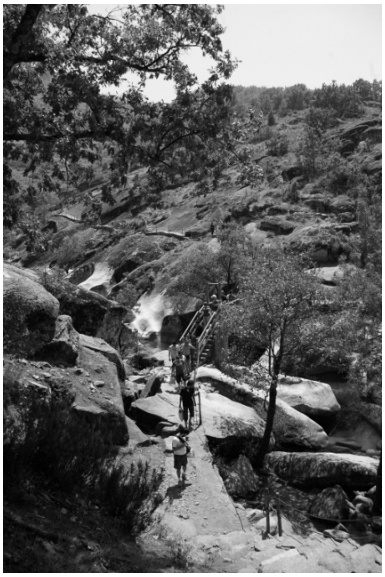
2.6.2 GPG サイン

Debconf9 開催前に DSA の脆弱性が発覚し、これにより新しい GPG 鍵を作成する必要がでてきました。新しく作成した鍵を使って開発者同士で鍵サインをする必要が出てきたため、今回の Debconf では GPG サインが活発に行われました。今回のポイントはこれまでの事務的な手続きを反省して、Debconf 開始に際してセッションを開き、チェックサムの検証を行って、あとは好きなときにちゃんと相互におしゃべりをしながらサイン交換しましょうということがとても重要だったと思います。名札に GPG 鍵リスト上で割り当てられた番号が書かれていて、やりとりを促進するようにもなっていました。おかげで我々は 29 日夜のサインパーティに出られないながらも、hacklab や食事どき、デイトリップの最中などいろいろな場面で多数サイン交換ができました。

武藤さん、岩松は 40 名ぐらいの参加者と交換することができました。

2.7 Daytrip

Debconf では一日、参加者で旅行をするというイベントがあります。今回の Debconf では 会場からバスで 2 時間ほど移動したところにあるエクストラマドゥーラ州が運営しているキャンプ場に移動し、そこから 1 時間ほどハイキングをしました。ハイキングの先には、自然でできたプール（Natural Pool）があり、そこで皆で泳いだり飛び込んだり、滝に打たれていました。一部溺れかけてた人もいたようですが、皆無事だったようです。4 時間ほどそこで遊んだ後、別の場所に移動し、みんなでビールを飲みました。そこにも川をせき止めたプールがあるのですが、皆のグランパである Joey が足をざっくり切ってしまうイベントが発生し、救急車が呼ばれるハプニングがありました。



2.8 Debconf11

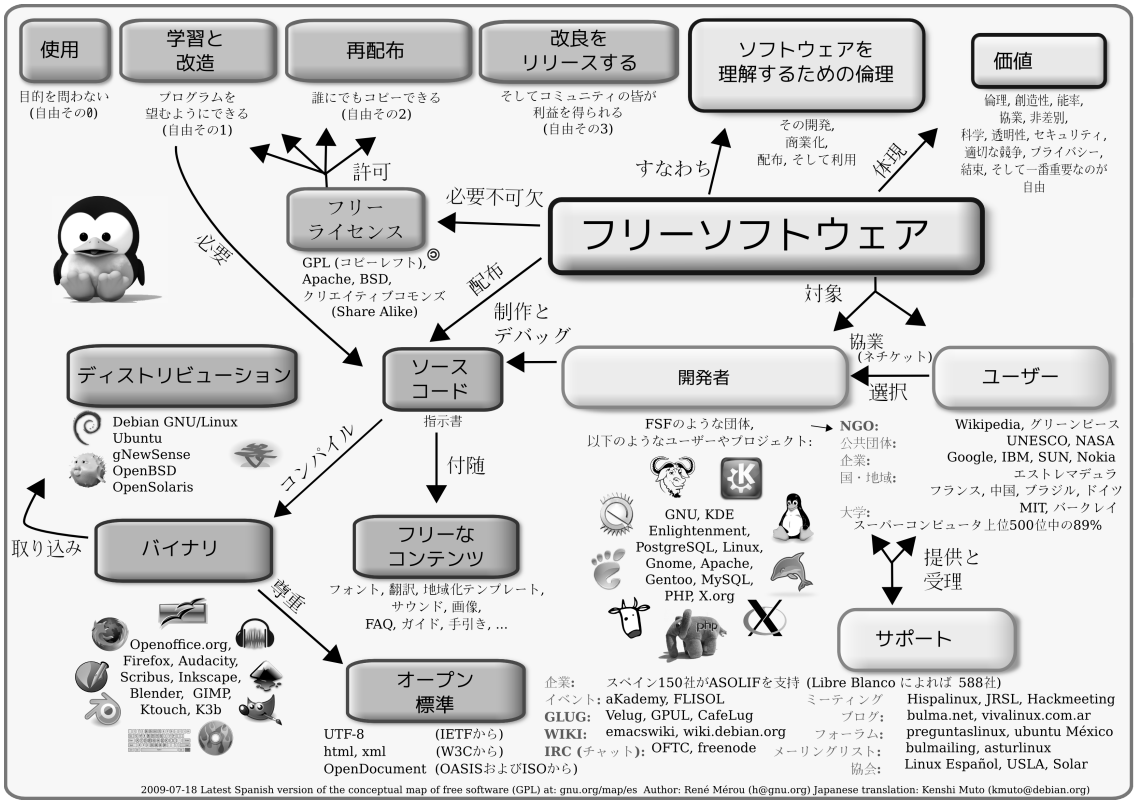
Debconf11 はボスニアとドイツが立候補しました。ドイツはまだ場所が決まっていないようですが、今年中に場所を確定させるようです。ボスニアはトランジットが面倒ですが、開催地は空港から数キロのところにあり、そんなに困らないように感じました。ドイツは Debian 開発者が一番多い国で、Debconf チーム (orga) のメンバも多いため、ドイツ濃厚な気がします。

2.9 今回の Debconf 参加によるハックの成果

2.9.1 武藤さん

ツアーコンダクターとして、マドリード観光や現地での皆さんの安全・食事探しに取り組みました。

- ドラゴンクエスト IX
  出発時には最初の村を出るところでしたが、移動時間中だけで遊んでいたにもかかわらず、成田に戻る飛行機ではラスボスと戦っていました。全滅してしまったのでもう少し経験を重ねたいと思います。
- フリーソフトウェアマップの翻訳
  スペインでフリーソフトウェアの活動を進めている GNU プロジェクトメンバーの Rene Merou に頼まれて、彼らの作っているフリーソフトウェアマップを日本語に翻訳しました。政府機関にフリーソフトウェア採用を働きかけるなど、活発な活動をしているそうです。



- ハードウェア互換性リストとハードウェアサポートページの友好関係構築
  Debian のハードウェアサポートの情報を収集して Wiki(http://wiki.debian.org/HardwareDatabase, http://wiki.debian.org/DeviceDatabase/PCI など) にまとめている Franklin Piat に会って、私の Debian HCL(http://kmuto.jp/debian/hcl/) とどのような協業ができるかを話し合いました。まずは彼の USB データベースの成果を生かして、lsusb コマンドを貼り付けることで USB のサポート情報を表示するように HCL を拡張できるかを検討することになりました。
- 国際化チームミーティング
  恒例の国際化チームミーティングは期間中 4 回開催され、Wiki や DDTP について議論がなされました。Squeeze ではリリースノートを Wiki ベースで制作・編集するというトライアルが実施されます。Wiki のように変動の激しいものを翻訳するための手法として、po フォーマットに変換してパラグラフ単位で作業できるような手法も実施予定です。Debian のパッケージ説明翻訳機構の DDTP についても、多数の改善提案が出されました。パラグラフ単位でのマッチングや fuzzy マーキング、一般翻訳者を統轄するコーディネータという役割および特権を用意して一括置換や管轄する翻訳メンバーへの一斉連絡といった機能が追加される見込みです。日本の DDTP 翻訳チームでも、いずれどなたかにコーディネータに立候補いただくことになると思われます。
- 講演レポート
  各講演内容については `http://kmuto.jp/d/index.cgi/travel/20090722-spain.htm`, `http://kmuto.jp/d/index.cgi/travel/20090724-spain.htm`, `http://kmuto.jp/d/index.cgi/travel/20090726-spain.htm`, `http://kmuto.jp/d/index.cgi/travel/20090728-spain.htm` に短いながら報告しています。詳細については Debian の関連メーリングリストや Debconf サイトで掲載予定のビデオなどを参照してください。

3 年ぶりの参加でしたが、新旧の友人たちと語らい、GPG サイン交換し、Debconf 初参加者に Debconf を体感してもらい、国際化チームミーティングで up-to-date な情報を得る、と当初の目的はいずれも達成できました。充実したカンファレンスだったと思います。

私自身は今後徐々に Debian に関する活動のペースを落としていくつもりなので (すでにだいぶ落ちてしまってはいますが)、特に DDTP 関係者の Debconf および国際化チームミーティングへの参加を望みます。

2.9.2 山根さん

兎に角、「参加することに意義がある」と言うことで参加してみました。初海外だったのでドキドキものです。色々と皆さんに助けていただきました。

- po-debconf 訳
  多少は fuzzy, unstranslated を潰しました。
- フォントパッケージのライセンス確認
  日本語フォントについて、何故かスペインについてから upstream に確認を取り始める私。「OFL だよ」「完全にフリーです、著作権も行使しません」との返答を得ました。パッケージにするのは description と defoma-hints ファイルの作成が待ってますが…
- 参加レポート
  これとは別にまた少しだけ書いておこうと思います。
- GPG keysign
  15、6 人ほどと鍵交換したはず。caff 便利なんですね。そのうち使い方とかまとめたいかも。
- java policy 訳 review
  帰りの飛行機で半分ほどをレビューしました。何故。

逆に課題としてもらってきたのは「英語英語英語〜」ですね。かなり精進が必要な模様です。

2.9.3 前田さん

数ヶ月溜まっていたタスク、問題を ToDo として取り組みました。

- MacBook 5,2 の使えないデバイスを使えるようにする
  所有している MacBook は 2008 later モデル (通称 MacBook 5,2) ですが、DebConf に行く前に以下のような問題を抱えていました。
  - 無線 LAN を使えない
    Broadcom bcm4322 が搭載されており、b43 で将来的にはサポートされるらしいのですが、現状では Broadcom が提供している STA ドライバを使用するしかありません。ただし、普段、vanilla kernel の最新 stable を使用していると、このデバイスドライバをビルドできない、という問題があります。スペインから Twitter でつぶやいていたら、Gentoo の松鵜さんに、Gentoo のパッチを使ってみては、と教えていただき、x86 用のパッチを参考に、amd64(x86_64) 用にコードを書き直したところ、正常に無線 LAN を使えるようになりました。
    実は Debian に既に Broadcom-sta ドライバ用のパッケージがあり、これを使うと上記の問題は既に解決されている、というオチもついてました。[*1]
  - 音が出ない
    サウンドデバイスは認識しているものの、音が出ない、という現象にハマっており、/etc/modprobe.d/alsa-base.conf の設定をしなおすことでシステムブート時に音が出るようになりました。
    が、これが非常にけたたましいビープ音で、イヤホンジャックにイヤホンを挿してもなりやまず、Hacklab1 内でかなり顰蹙 (ひんしゅく) を買ってしまいました…。orz
    帰国してから再設定したのですが、再び音が出なくなり、四苦八苦中です。
  - iSight が対応していない
    isight-firmware-tools が、MacBook 購入時 (2009 年 5 月末) では対応していなかったのですが、久しぶ

[*1] broadcom-sta-common, broadcom-sta-source パッケージ。i386 用にはバイナリパッケージもあるようです。

りに実行してみたら何もすることなくあっさりファームウェアを抽出でき、使えるようになりました。
  - ACPI を有効にすると起動できない
    この問題は未だ解決しておらず、ACPI を無効にするためバッテリー駆動の場合、残量が分からないとか、正常に電源を切れないため reboot ができず必ず shutdown しないといけないとか、ハイバネーションを行うと、復帰時に ACPI を有効にしたときと同様に画面がブラックアウトして固まってしまう、という問題があります。
    DebConf で MacBook を使っている人に聞いてみようかと思っていたのですが、同じ世代の MacBook を使っている人が少なく、持っていても Mac OS X を使っているので、断念しました。
    ACPI の ML で聞いてみる予定です。
- Java Policy の翻訳
  4 月の Debian 勉強会のネタとして、Java Policy を読んでみましたが、中途半端なところで翻訳を中断していました。再開するにあたり、PO 形式で整形し直し、未翻訳の部分の翻訳を完了し、debian-doc に投稿しました。
  レビューコメントをやまねさんに頂いたのですが、帰国してから飼いはじめたネコに夢中で再び停滞しているので、今回の勉強会後に確認し、修正後、Java Policy の ML への連絡及び、java-common パッケージの BTS を行う予定です。
- GanttProject のパッケージ化
  ソースコードの入手方法が分からず、放置していたのですが、Subversion のリポジトリがあることに気づいたので、ドキュメントに沿ってビルドしてみたところ、OpenJDK では正常に動くことを確認しました。
  他の Java 開発環境でのビルド、動作確認後、deb パッケージ化を行う予定です。
- Termtter のパッケージ化事前調査
  jugyo さんが中心になって開発されている Twitter のターミナルクライアントがあります。iceweasel の extention である twitterfox の使い勝手がいまいちなのと、ちょうど DebConf9 中に動作がおかしくなり、有効にしていると iceweasel ごと busy になってしまうので、切り替えることにしました。使い勝手がよければ、deb パッケージにしてしまおうと考え、KVM 環境で検証してみたところ、rubygems で必要なライブラリを導入し、既に deb パッケージで導入済みのライブラリを無視してしまいます。
  今後の予定としては、その当たりの解決と、gems で導入されている deb パッケージになっていないライブラリ自体のパッケージ化の検討を行うつもりです。
- おまけ 1。最近中古で買ったスーパーマリオブラザース DS をクリアしました。クッパに息子がいるとは。またクッパはゾンビなってしまったのですね。
- おまけ 2。最近、帰国後から飼い始めたネコですが、帰国後に引き取りに行くことが決まっていたので、事前にネコの健康本と、ネコの気持ちが分かる本を買っていきました。、帰宅後、熟読して帰ったらヨメにネコ博士と呼ばれるようになりました。

セッションにはいくつか参加しましたが、知らない内容を勉強するために出てみても、英語の聞き取りがほとんど出来ないので、言っていることが分からないという問題にぶち当たりました。そういう場合は、資料が頼りなのですが、事前に配布されていないセッションも多く、おまけにスライドの文字が小さくて前の方の席で眼鏡をかけても見えないようなセッションでは、非常に大変でした。

2.9.4 岩松

岩松は主に sh4 アーキテクチャ向けの開発を行いました。

- buildd hack
  Debian / SH4 の buildd が動きはじめました。これにより、SH4 向けバイナリが debian-ports.org が取得できるようになります。パッケージ作成の進捗は `http://www.debian-ports.org` から参照することができます。

また、armel buildd メンテナである Riku と組み込み向け CPU で発生する問題をどのようにフックしていくのか、意見交換を行いました。

- debian kernel
  tbm [*2] を捕まえて、debian kenrel config ファイルのハックをしました。sh4 向けカーネルサポートを一通りできました。
- sh4 向けクロスコンパイラのサポートの打ち合わせ
  gcc のクロスコンパイラサポートをしている zumbi と会話し、sh4 と uclibc のクロスコンパイラサポートの話をしました。gcc-4.4 ではうまく動作しないので修正したり、クロスコンパイルに必要なパッケージを提供したりしていました。
- defoma
  defoma を ITA したままほったらかしにしていたので、CPAN admin をつついて、バグのある CPAN の乗っ取りを再開しました。CPAN はやりとりが面倒くさいので新しいソフトウェアを作成したほうが早そうです。
- Linux kernel/ U-boot
  こっちに来ている間にもパッチが飛び交っているのでメールのチェックとパッチの査読をしていました。
- speedstep-centrino 自分の使ってる MacBook は CoreDuo なので、speedstep-centrino が使えるはずなのですが、モジュールのロードに失敗します。CPU マニュアルを斜め読みして対応してみました。

2.10 次回の Debconf

次回の Debconf10 は 米国のニューヨークで開催される予定です。一部の日本人が日本での開催を再度目論んでいるようです。スポンサーとか会場情報募集中。

[*2] 元 DPL, arm のカーネルメンテナ

3 Debian GNU/kFreeBSD をインストールしてみよう

山本 浩之

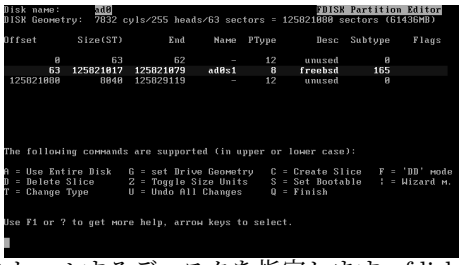
今年 4 月 5 日に正式に Debian archive に入った Debian GNU/kFreeBSD ですが、現時点で最新インストーラである debian-20090117-kfreebsd-i386-install.iso (2009/1/17 製) を使ってインストールしてみました。

英文ドキュメント `http://glibc-bsd.alioth.debian.org/doc/`

```
wget http://glibc-bsd.alioth.debian.org/install-cd/kfreebsd-i386/current/debian-20090117-kfreebsd-i386-install.iso
```

このイメージにある kernel は、FreeBSD 7.1R のものですが、インストール後、7.2R の kernel にアップグレード可能です。

今回は VMware の仮想ディスクにインストールを試みました。インストーラは、現在のところは、FreeBSD 用のものを一部改造したもののようです。インストーラを CD イメージから起動します。とりあえず Express (または Custom) でインストールしてみましょう。

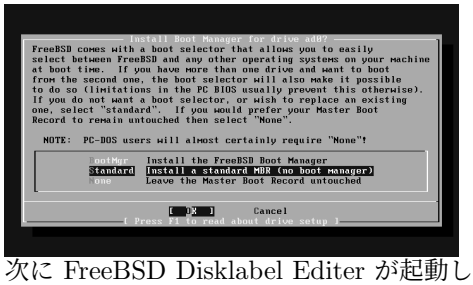


まず最初にすることは、Linux と同様、インストールするディスクを指定します。fdisk が起動しますが、ここで注意しなくてはならないのは、Linux とはディスクの指定の仕方が違うことです。FreeBSD では、hda1 にあたるものは、ad0s1 で、スライスと呼びます。FreeBSD では伝統的に、使用する領域全部を先にスライス (ad0s1) として確保し、その中にディスクラベルをつけることによって小分けして (ad0s1a、ad0s1b など)、それぞれのマウントポイントを作ります。Linux の場合のパーティションの設定に近いものはディスクラベルかもしれません。とりあえず "A" で、空き領域全部指定しましょう。

次にブートローダの書き込みですが、デフォルトでは FreeBSD Boot Manager と言う FreeBSD 専用のプログラムが使用されます。もし既存の grub が使いたい場合には、ここでは書き込まず (None を指定)、インストール後に、既存の grub が使っている/boot/grub/menu.lst に

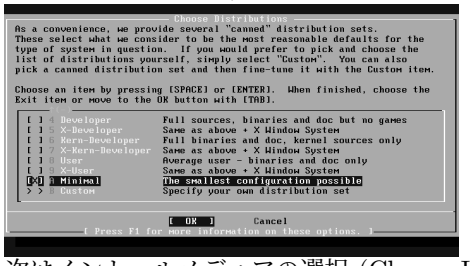
```
title Debian GNU/kFreeBSD
root (hd0,0,a)
kernel /boot/loader
```

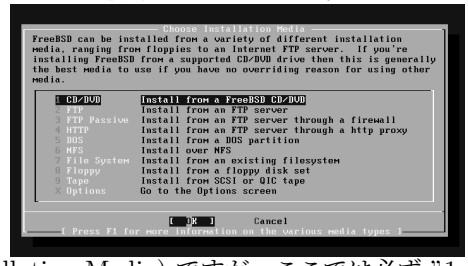
とか書けば良いそうです。仮想ディスクなどに対してのインストールで、マルチブートの必要が無い時は "Standard" を選んでください。FreeBSD Boot Manager を使ってマルチブートしたい場合にのみ "BootMgr" を選んでください。



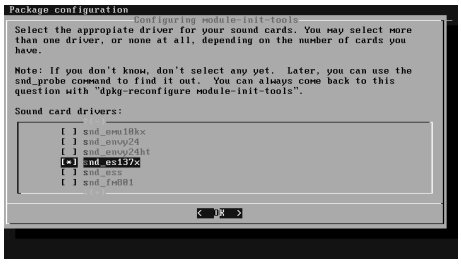
次に FreeBSD Disklabel Editer が起動し、スライスの中に、さらにマウントポイントとしていくつもディスクラベルをつけます。それぞれ ad0s1a ad0s1b ... などとなります。”/” のため、少なくとも一つはディスクラベルをつける必要があります。もし、”/usr” とか ”swap” とかを分けたいときにはここで選びます。”A” で、自動的に割り振ることも可能です。

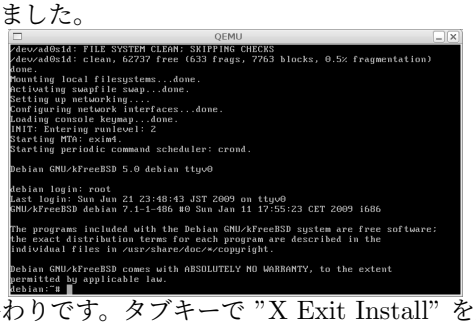
そして Distribution の選択 (Choose Distribution) ですが、ここでは必ず ”A Minimal” を選択してください。ここで色々な選択肢が出てきますが、これは元の FreeBSD のインストーラだったころの名残りで、これは Debian のインストーラですから、FreeBSD の Distribution は全く収録されていません。





次はイントールメディアの選択 (Choose Installation Media) ですが、ここでは必ず ”1 CD/DVD” を選択してください。その他の選択肢は FreeBSD のインストーラだったころの名残りです。自分の環境にあわせ、”cd0” (SCSI) または ”acd0” (ATAPI) を選べば、インストールの始まりです。途中で、”Alt-F3” を押すよう指示がありますから、それに従うと、Debian GNU/Linux でお馴染みな、タイムゾーンの設定とか popularity-contest の設定とかができます。Debian GNU/kFreeBSD 特有なものは、module-init-tools の設定で、これは FreeBSD カーネルのモジュール名を選ばなければなりません。ネットワークカードモジュールとサウンドカードモジュールとその他のモジュールが出てきますが、詳しくは FreeBSD 本家のドキュメントを見てください。私の場合は VMware ですので、サウンドカードモジュールの ”snd_es137x” のみ選びました。





最初の画面に戻ってきたらインストールは終わりです。タブキーで ”X Exit Install” を選び、再起動して下さい。

最初は root ユーザのみがパスワード無しで出来ています。ユーザ名 root を入力すると、パスワードをきかれずにログインできます。

root でログインしたら、まずは passwd コマンドでパスワードを変更しましょう。

```
passwd
```

次にネットワークの接続をします。Linux の eth0 にあたるものは le0 (VMware の場合) または ed0 (Qemu の場合) のようです。dhcp3-client パッケージは既にインストール済みのはずですから、DHCP 環境[*3]の人は /etc/network/interfaces を編集しなくても

```
# dhclient3
```

[*3] qemu の場合デフォルトでは dhcp で IP アドレスが取得できます

でネットワークに繋がるはずです。
固定 IP の人は、以下を参考にして、それぞれのファイルを編集して下さい。

```
##/etc/network/interfaces の例
auto lo0
iface lo0 inet loopback
## Static network
auto le0
iface le0 inet static
    address 192.168.0.3
    network 192.168.0.0
    netmask 255.255.255.0
    gateway 192.168.0.1
```

```
##/etc/resolv.conf の例
nameserver 192.168.0.1
```

その後、

```
# ifup le0
```

で、ネットワークに繋いで下さい。
インターネットに繋がったら、まず keymap の設定のため、

```
# apt-get update
# apt-get install kbdcontrol
```

をして、console の keymap を選んでください。(console-data は使えません)
インストール後は、Debian GNU/Linux とどこが違うのか分からない程、まさに Debian です。

3.1 参考文献

- Installing Debian GNU/kFreeBSD `http://glibc-bsd.alioth.debian.org/doc/`: インストール方法が詳しく記載されています。

4 【CD/DVD/USB メモリ】Debian JP 版 Debian Live を作るよ【netboot も】

のがたじゅん

4.1 はじめに

OS をハードディスクにインストールせずメディアから直接起動して使う「ライブシステム」が知られて久しいですが、Debian にも Debian Live というライブシステムがあります。今回は Debian Live 作成ツールの live-helper を使って、Debian JP 版 Debian Live や関西 Debian 勉強会の配布 DVD を作成する中で、自分好みの Debian ライブシステムを作成する方法やコツなどを解説します。

4.2 Debian Live Project とは

Debian Live の作成の前に、Debian Live Project について紹介します。

Debian Live Project は、ライブシステム作成のためのフレームワーク live-helper と live-initramfs などのライブシステムにまつわるユーティリティを開発するプロジェクトで、Debian の公式サブプロジェクトです。

既存のライブシステム・ディストリビューションとの違いは、ライブシステムを作ってリリースすることより、フレームワークとしてのツールの制作に重きを置いているところでしょうか。

live-helper を使って作られた Debian ライブシステムは Debian Live と呼ばれ、Lenny から同時にリリースされています。

4.3 Debian Live の特徴

Debian Live は、既存のライブシステム・ディストリビューションの欠点を解消するように作られており、以下の特徴を持っています。

- 一つのディストリビューションのみで完結。
- カーネルも含めてパッチを当てたりなど特別なパッケージを使わない。
- リマスタリングではない新しい環境からシステムを作ることができる。
- CD、DVD、USB メモリ起動のシステムだけでなく、ネットワークやインターネット越しに起動するシステムを作ることができる。
- 複数のアーキテキクチャをサポート。
- Debian Installer を含めることができる。

Debian Live は Debian のみで作成できるようにデザインされていますが、自作のパッケージや独自の Apt リポジトリのパッケージを追加して作ることも可能です。

アーキテキクチャについては、現在、正式に対応しているのは i386、amd64、powerpc だけですが、基本的に Debian がサポートしているアーキテキクチャには対応する予定です。サポートしているモードも Debian だけでなく、emdebian や ubuntu にも対応しています。(sid 以上で –mode ubuntu と指定すると不完全ながらも ubuntu live ができます。)

Debian Installer については通常の netinst や businesscard のインストーラに加え、Debian Live のパッケージをそっくりそのままインストールする live もサポートされています。また、計画段階ですが、Debian Live 上から直接 Debian をインストールするためのインストーラの制作も予定されています。

4.4 live-helper について

live-helper とは、Debian Live を作成するためのファイル名の頭に「lh_」とついたシェルスクリプト群です。群というだけあってスクリプトは多数ありますが、この中で Debian Live 作成に使用するコマンドはたった 3 つ。設定のための「lh_config」、作成のための「lh_build」、作業ディレクトリをクリーンナップする「lh_clean」だけです。

残りのスクリプトは、3 つのスクリプトの中から設定に応じて適宜呼ばれるので、通常、意識する必要はありません。

4.5 Debian Live の最初の一歩

それでは live-helper を使った Debian Live の作成の基本について解説します。

4.5.1 live-helper のインストール

live-helper は、すでに Debian のリポジトリに用意されているので、aptitude などを使ってインストールします。パッケージインストールについての注意ですが、live-helper パッケージに Suggests や Recommends されたパッケージを使用する場面が多いので、特に事情が無ければ、すべてインストールしておいてください。

live-helper のインストール

```
# aptitude --with-recommends install live-helper
```

4.5.2 作業の準備

Debian Live 作成のための作業ディレクトリを作成します。ここでは live-work と名づけて作りました。

作業ディレクトリの作成

```
$ mkdir live-work
```

Debian Live の設定ファイルや作成作業のファイルは、すべてカレントディレクトリに置かれます。よく使うディレクトリで Debian Live 作成作業をおこなうと、これらのファイルと通常のファイルが混ざって収集がつかなくなるので、作業の前には専用のディレクトリを用意しましょう。

4.5.3 まず Debian Live を作ってみる

設定には lh_config コマンドを使います。作業ディレクトリに降りて lh_config と入力します。

```
$ cd debian-live/
$ lh_config
```

config と scripts ディレクトリが作成されたはずです。ディレクトリの意味については後ほど説明するので、まずは Debian Live を作成してみましょう。「sudo lh_build」と入力します。(lh_config コマンド以外は管理者権限が必要になるので、コマンドの前に sudo をつけて実行します。)

```
$ sudo lh_build
```

マシンとネットワークの状況にもよりますが 15 分〜30 分ほどで binary.iso、binary.list、binary.package というファイルができているはずです。他には binary、cache、chroot というディレクトリができています。

できていなければ「sudo lh_clean」と入力し、作業途中のディレクトリを消去してから、もう一度試してみてください。

```
$ ls -la
drwxr-xr-x  6 jun  jun        296 2009-06-24 16:24 .
drwxr-xr-x 10 jun  jun        320 2009-06-24 16:08 ..
drwxr-xr-x  2 root root       640 2009-06-24 16:24 .stage
drwxr-xr-x  6 root root       176 2009-06-24 16:24 binary
-rw-r--r--  1 root root 132192256 2009-06-24 16:24 binary.iso
-rw-r--r--  1 root root      2171 2009-06-24 16:24 binary.list
-rw-r--r--  1 root root     11123 2009-06-24 16:23 binary.packages
drwxr-xr-x  6 root root       184 2009-06-24 16:08 cache
drwxr-xr-x 20 root root       600 2009-06-24 16:24 chroot
drwxr-xr-x 22 jun  jun        936 2009-06-24 16:08 config
```

4.5.4 イメージファイルを確認する

生成されたファイルの中に binary.iso というファイルがあります。これが Debian Live の CD イメージファイルです。それでは起動テストをおこないますが、CD-R などに書き込んでのテストはマシンを再起動しなければいけませんし、ライトワンスのメディアを使うのは環境にも良くないので、仮想マシンを使って確認します。

例では qemu を使いましたが、KVM や VirtualBox、VMware Player など好きな仮想化ソフトを使ってかまいません。qemu を使う場合、そのままの状態では遅いので、あらかじめ高速化カーネルモジュールの kqemu を組み込んでおいてください。

kqemu を組み込む

```
# m-a a-i kqemu
# modprobe kqemu
```

qemu 上で Debian Live を実行する

```
$ qemu -cdrom binary.iso -boot d -m 256
```

qemu 上で Debian Live が起動したでしょうか？画面は白黒のコンソール画面、キーボードは英語配列。想像していたものと、だいぶかけ離れた画面が現れたと思います。

これが最小の Debian Live です。ここから自分なりの設定を加えて、自分オリジナルの Debian Live を作り上げます。

4.6 Debian Live の設定

さて、これから本格的な Debian Live の設定に入りますが、Debian Live が作成される手順を知っておくと、より的確に設定することができるので説明します。

Debian Live の作成には lh_build コマンドを使いますが、lh_build は 4 つのコマンドを呼び出しそれぞれの作業をおこないます。その 4 つのコマンドは以下になります。

1. debootstrap によるベースシステムのインストール (lh_bootstrap)
2. ベースシステムに chroot して必要なソフトのインストールや設定をおこなう (lh_chroot)
3. 作成されたシステムを一つのファイルにまとめ、起動できるバイナリイメージを作成 (lh_binary)
4. 作成されたバイナリイメージにソースが必要ならば、ソースをまとめたイメージを作成 (lh_source)

このように bootstrap → chroot → binary → source の順に作業が進み、それぞれのステージにおいて独自の設定をすることができます。

設定ファイルは config ディレクトリ以下にありますが、common、bootstrap、chroot、binary、source の 5 つのファイルは lh_config から設定するので、直接編集しません。

それ以外のディレクトリでは、chroot ステージに関係したものは「chroot_」、binary ステージに関係したものは「binary_」と、それぞれの状態を表すプリフィクスがついているので、参考にしながら場面に応じた設定をしていきます。

4.6.1 設定と作成の基本

Debian Live の設定は lh_config コマンドを使って行います。lh_config –help と入力してみましょう。

```
$ lh_config --help
```

ものすごい数の設定が出てきましたが、すべてを設定する必要はありません。設定をしなければ基本の設定が使われるので必要な箇所のみ変更していきます。

```
$ lh_config \
        --binary-images iso \
        --distribution lenny \
        --language ja \
        --bootappend-live "quiet locale=ja_JP.UTF-8 keyb=jp kmodel=jp106" \
        --mirror-bootstrap "http://ftp.jp.debian.org/debian/" \
        --mirror-chroot "http://ftp.jp.debian.org/debian/" \
        --mirror-chroot-security "http://security.debian.org/" \
        --mirror-binary "http://ftp.jp.debian.org/debian/" \
        --mirror-binary-security "http://security.debian.org/"
```

上から順に説明します。

–binary-images は生成するバイナリイメージの種類を指定します。iso 以外にも usb-hdd など指定できます。–distribution にはディストリビューションを指定。lenny を指定しています。squeeze は現在カーネルパッケージの不整合があるので作れません。–language は言語です。Iceweasel や OpenOffice.org のように言語別パッケージがあるときの判断に利用されます。

–bootappend-live は Debian Live 起動時のブートパラメータを指定します。ここではカーネルのメッセージ出力を抑制する「quiet」と、ロケールとキーボードを日本語の設定にしています。quiet 以外は Debian Live 独自のパラメータです。パラメータ一覧は、live-initramfs パッケージの/usr/share/doc/live-initramfs/parameters.txt を参照してください。

–mirror は Apt の取得先を指定します。ミラーは、それぞれのステージで設定できますが、特別な事がない限り分けて指定する必要はないので、ftp.jp.debian.org と security.debian.org を設定しています。

とても長かったですが、これを毎回設定するのは大変です。毎回設定しないようにするには、どうしたらいいのでしょうか。

lh_config を最初に実行した際、設定を保存する config ディレクトリのほかに、scripts ディレクトリが生成されていたことを覚えているでしょうか。その scripts ディレクトリに設定のための config をスクリプトとして置きましょう。

scripts/config

```
#!/bin/sh

MIRROR_DEBIAN="http://ftp.jp.debian.org/debian/"
MIRROR_SECURITY="http://security.debian.org/"

BOOTOPTION_LIVE="quiet locale=ja_JP.UTF-8 keyb=jp kmodel=jp106"

lh_config noautoconfig \
        --binary-images iso \
        --distribution lenny \
        --language ja \
        --bootappend-live "${BOOTOPTION_LIVE}" \
        --mirror-bootstrap ${MIRROR_DEBIAN} \
        --mirror-chroot ${MIRROR_DEBIAN} \
        --mirror-chroot-security ${MIRROR_SECURITY} \
        --mirror-binary ${MIRROR_DEBIAN} \
        --mirror-binary-security ${MIRROR_SECURITY}
        ${@}
```

この scripts/config スクリプトは、lh_config コマンドを実行したとき、scripts/config スクリプトが存在すれば、再帰的に呼び出され実行される仕組みになっています。

これを見て **わざわざ手間のかかることをしているのはなぜ?**と思った方もいると思います。それには**クリーンな環境からのビルド**と**ビルドの自動化**の二つの理由があります。

「クリーンな環境からのビルド」ですが、live-helper で生成した設定ファイルは今は何も変わりませんが、この先バージョンが上がった場合どうでしょう? オプションが廃止されたり、新しいオプションが追加されるかもしれません。古いバージョンの設定ファイルを使いつづけていると、それらに気づかないまま対応できない以外にも、無用なトラブルを呼び込むかもしれません。

それらを避けるためにも毎回 config ディレクトリを消去して新たに設定を生成する必要があります。

もう一つ **ビルドの自動化**ですが、ライブシステムでは、同一内容で設定が少し違うものが欲しいことがあります。たとえば CD イメージ版と USB メモリ版などです。

そのとき、毎回オプションを指定して作ることは効率が悪いので、共通する設定はスクリプト化しておき、変更点のみ指定して 2 回呼び出せば、クリーンかつ必要なイメージを作成することができます。

scripts/config スクリプトに続いて、lh_build コマンドから呼び出される scripts/build スクリプトと、lh_clean コマンドから呼び出される scripts/clean スクリプトも作っておきましょう。

scripts/build スクリプトでは、ビルド作業のログと生成されるイメージをわかりやすくするためファイル名に作業時間を含めるようにし、scripts/clean スクリプトは、config ディレクトリ内で空のディレクトリも消去するようにしています。

scripts/build

```
#!/bin/sh

IMAGE_PREFIX=debian_live-binary
BUILDDATE=‘date +%Y%m%d%H%M%S‘

lh_build noautoconfig 2>&1 | tee ${IMAGE_PREFIX}-${BUILDDATE}.buildlog

# rename files
if [ -f binary.iso ]; then
   mv binary.iso ${IMAGE_PREFIX}-${BUILDDATE}.iso
elif [ -f binary.img ]; then
   mv binary.img ${IMAGE_PREFIX}-${BUILDDATE}.img
fi
[ -f binary.list ] && mv binary.list ${IMAGE_PREFIX}-${BUILDDATE}.list
[ -f binary.packages ] && mv binary.packages ${IMAGE_PREFIX}-${BUILDDATE}.packages
```

scripts/clean

```
#!/bin/sh

lh_clean noautoconfig --all ${@}

# Remove generated files
rm -f config/binary config/bootstrap config/chroot config/common config/source

# Remove empty directories in config tree
if ls config/*/ > /dev/null 2>&1
then
    rmdir --ignore-fail-on-non-empty config/*/
fi

if [ -d config ]
then
    rmdir --ignore-fail-on-non-empty config
fi
```

これで lh_config と入力すると、いつでも自分の基本設定の状態から作成することができます。sudo lh_build や sudo lh_clean と入力するとログをとりながらビルドしたり、クリーンナップもできます。

さて、自動化といえば Make。ということで簡単ですが Makefile も用意してみました。sudo lh_build なんて長く打たなくても make と入力するだけでビルドできますし、ほんの少し設定を変えるときも Makefile の中で処理することができるので、とても楽になりました。

Makefile

```
all: config build

config: clean
        lh_config

build:
        sudo lh_build

clean:
        sudo lh_clean

distclean: clean
        sudo lh_clean --purge
        sudo rm -f *.iso *.img *.list *.packages *.buildlog *.md5sum
```

もう一つついでに、git で live-helper のレシピを管理するととても楽ですが、その場合、以下のような「.gitignore」を作成しておくとよいでしょう。

```
*.buildlog
*.img
*.iso
*.list
*.md5sum
*.packages
*~
.*~
.stage/
binary/
cache/
chroot/
config/binary
config/bootstrap
config/chroot
config/common
config/source
```

4.7 Debian Live のカスタマイズ

作成環境ができあがったので、具体的なカスタマイズについて述べます。

4.8 パッケージの追加

自分好みの Debian Live にするために最初に始めることはパッケージの追加でしょう。パッケージを追加するにはいくつか方法はありますが順番に説明します。

4.8.1 パッケージを追加する

APT リポジトリにあるパッケージを追加するには–packages オプションを使います。パッケージの指定方法は、パッケージ名をスペースで区切って並べていきます。

```
$ lh_config --packages "PACKAGE PACKAGE2 ···"
```

自作パッケージを追加するには、config/chroot_local-packages/ディレクトリに deb パッケージを置き、–packages オプションでパッケージ名を指定します。

4.8.2 パッケージリストで追加する

–packages オプションでは、まとまった数のパッケージを追加するには、いくつもパッケージ名を並べないといけないので不便です。よく使われるデスクトップ環境などのパッケージリストはデフォルトで用意されているので、–packages-lists オプションを使って指定します。

パッケージリストの一覧は/usr/share/live-helper/lists/をご覧ください

```
$$ lh_config --packages-lists "gnome"
```

自分で追加したいパッケージ名を並べたパッケージリストを config/chroot_local-packageslists/ディレクトリに用意し、–packages-lists オプションで指定することもできます。

「FILE」という名前のパッケージリストを作成

```
$ vi config/chroot_local-packageslists/FILE

$ cat  config/chroot_local-packageslists/FILE
lv manpages-ja nkf
iceweasel-l10n-ja
openoffice.org-help-ja openoffice.org-l10n-ja
ttf-kochi-gothic ttf-kochi-mincho ttf-sazanami-gothic ttf-sazanami-mincho ttf-vlgothic
uim uim-applet-gnome uim-prime uim-qt uim-qt3
```

–packages-lists オプションで「FILE」を指定します。

```
$ lh_config --packages-lists "FILE"
```

パッケージリスト書式

別のパッケージリストを含める方法

```
#include <gnome>
iceweasel
```

アーキテクチャで分岐する方法

```
#if ARCHITECTURE amd64
ia32-libs
#endif
```

セクションで分岐する方法

```
#if SECTIONS contrib non-free
vrms
#endif
```

4.8.3 APT リポジトリを追加する

APT リポジトリを追加するには、config/chroot_sources/ディレクトリに Apt-line を書いたファイルと、パッケージの署名を検証する GPG 鍵ファイルを置きます。

Apt-line を書いたファイル名は chroot ステージと binary ステージで別々に指定することができ、chroot ステージの場合には拡張子を **.chroot**、binary ステージでは拡張子を **.binary** にします。GPG 鍵ファイルは拡張子を **.gpg** にしておきます。

```
restricted-debian_multimedia.chroot
restricted-debian_multimedia.chroot.gpg
restricted-debian_multimedia.binary
restricted-debian_multimedia.binary.gpg
```

キーリングパッケージを指定する

GPG 鍵がキーリングパッケージとして配布されている場合は、–keyring-packages オプションにキーリングパッケージを指定します。

```
$ lh_config --keyring-packages "debian-archive-keyring debian-multimedia-keyring"
```

4.8.4 ファイルを追加する

パッケージではなく、ファイルそのものを追加する場合は config/chroot_local-includes/ディレクトリにファイルを置きます。ファイルを置く場合の注意としては、ディレクトリ構造がそのままコピーされるので、/usr/local/bin/hoge というファイルを置く場合は、config/chroot_local-includes/ディレクトリをトップに見立て usr/local/bin/ディレクトリを作り、その中に hoge を置きます。

/usr/local/bin/hoge を置く場合のディレクトリ構造

```
$ ls chroot_local-includes/usr/local/bin/
hoge
```

4.8.5 カーネルパッケージを追加する

カーネルパッケージを指定してインストールするには、–linux-packages オプションと–linux-flavours オプションを設定します。

```
$ lh_config --linux-packages "linux-image-2.6 aufs-modules-2.6 squashfs-modules-2.6" --linux-flavours "686"
```

カスタムカーネルパッケージを追加する

カスタムカーネルパッケージを追加するには、カーネルのパッケージ以外にも、ファイルシステムを透過的に重ねる aufs/unionfs パッケージ、Kernel 2.6.28 以前のカーネルで squashfs を使うなら squashfs パッケージなどを用意する必要があります。

1. 用意したパッケージを config/chroot_local-packages/ディレクトリに置く。
2. –linux-packages オプションに”none”、–linux-packages オプションに用意したパッケージそれぞれを書く。–linux-flavours オプションにはパッケージ名のカーネルバージョン以降を書く。

```
$ lh_config --linux-packages "none" --linux-packages \
  "linux-image-2.6.26.6-rt11 aufs-modules-2.6.26.6-rt11 squashfs-modules-2.6.26.6-rt11" --linux-flavours "rt11"
```

4.9 設定のカスタマイズ

パッケージを追加したなら、次は設定を追加しましょう。設定の追加にもさまざまなパターンがあるので、順に紹介します。

4.9.1 シェルスクリプトを実行する

Debian Live で設定変更のため一番使われる手法は、シェルスクリプトを実行することです。パッケージのインストールから設定の書き換え、不要ファイルの除去など、ほぼなんでも行うことができます。

シェルスクリプトを実行するには、config/chroot_local-hooks/ディレクトリに実行させたいシェルスクリプトを置きます。

スクリプト例 (Iceweasel の初期設定を変更する)

```
#!/bin/bash
set -e

ICEWEASEL_PREFS=/etc/iceweasel/profile/prefs.js

cat << _EOL_ >>${ICEWEASEL_PREFS}

/* Debian Live tune */
user_pref("browser.cache.disk.parent_directory","/tmp");
user_pref("browser.cache.disk.capacity", 5000);
user_pref("browser.startup.homepage", "http://www.debian.or.jp/");
_EOL_
```

4.9.2 パッチを当てる

設定の変更にはシェルスクリプトを実行して sed で書き換えてもいいですが、パッチを当てたほうが早い場合もあります。パッチを当てるには chroot のトップから patch -p1 で当てられるように作ったパッチファイルを config/chroot_local-patches/ディレクトリに置きます。

パッチファイル例 (/etc/dhcp3/dhclient.conf を変更する)

```
*** chroot/etc/dhcp3/dhclient.conf.orig 2009-04-12 23:03:36.000000000 +0900
--- chroot/etc/dhcp3/dhclient.conf      2009-04-12 23:05:05.000000000 +0900
***************
*** 23,30 ****
       netbios-name-servers, netbios-scope, interface-mtu,
       rfc3442-classless-static-routes;
 #require subnet-mask, domain-name-servers;
! #timeout 60;
! #retry 60;
 #reboot 10;
 #select-timeout 5;
 #initial-interval 2;
--- 23,30 ----
       netbios-name-servers, netbios-scope, interface-mtu,
       rfc3442-classless-static-routes;
 #require subnet-mask, domain-name-servers;
! timeout 0;
! retry 0;
 #reboot 10;
 #select-timeout 5;
 #initial-interval 2;
```

4.9.3 debconf の設定をする

/etc にある設定などはシェルスクリプトやパッチで書き換えることができますが、debconf の設定は db に納められているので直接書き換えることはできません。その場合は config/chroot_local-preseed/ディレクトリに debconf の設定を置いて設定します。debconf の設定は、debconf-utils パッケージに納められている deboconf-get-selections を使って読み出します。

FUSE(Filesystem in Userspace) を使うためにユーザーを fuse グループに登録するには

1. debconf の設定を取り出して config/chroot_local-preseed/user-default-groups.preseed に書き出す

```
$ sudo debconf-get-selections | grep user-default-groups > config/chroot_local-preseed/user-default-groups.preseed
```

2. fuse グループを追加

```
user-setup      passwd/user-default-groups      string  audio cdrom dialout floppy video plugdev netdev powerdev scanner fuse
```

4.9.4 ブートローダーについて

ブートローダーについては i386/amd64 では syslinux と grub が選択できます。

ブートローダーのスプラッシュ画面を変更するには、syslinux の場合は config/binary_syslinux/ディレクトリに、grub の場合は config/binary_grub/ディレクトリに設定を置きます。

grub についてですが、usb-hdd では grub のインストールがまだサポートされていないので、syslinux しか使うことができません。誰か書いてみませんか？

4.9.5 スプラッシュスクリーンについて

最近のディストリビューションは起動時にスプラッシュスクリーンを表示していますが、Debian Live でも可能です。splashy か usplash パッケージをインストールして、–bootappend-live オプションに”vga=788 splash”と加えてください。vga の部分はフレームバッファの解像度なので適宜変更してください。

4.9.6 作成されるファイルとディレクトリについて

lh_config や lh_build が作成するファイルとディレクトリは以下のようになります。

表 2　Debian Live のディレクトリ

.stage/	進行状況を示すフラグが納められる
config/	設定が納められる
scripts/	作成のための config、build、clean スクリプトを置く
cache/	パッケージなどをキャッシュされる
chroot/	chroot イメージの作業ディレクトリ
binary/	binary イメージの作業ディレクトリ

4.10 その他、積み残したことなど

4.10.1 Debian Live を使っていてデータを保存するには

出来上がった Debian Live を使っていて、データを保存しておきたいことがあります。

その場合は、全体を保存するには live-rw、ホームディレクトリのみを保存するなら home-rw というラベル名で、ext2/3 でフォーマットしたパーティションをあらかじめ作成しておき、起動時、ブートパラメータに persistent をつけて起動すると自動的にそれぞれのパーティションをマウントします。

専用パーティションを用意しない場合は、live-snapshot コマンドを使うとデータが保存できるようなのですが、persistent をつけてもなぜか読み込んでくれないので、誰か live-snapshot の使い方を知っていたら教えてください。

4.10.2 live-helper と Debian Live 関連パッケージ

Debian Live 関連するパッケージについては以下のようになります。

Debian 公式パッケージ

- live-helper: Debian Live を作成するためのスクリプト。
- live-initramfs: Debian Live が起動する際に使われる initramfs の中で動くスクリプト。
- live-magic: GUI で Debian Live を作成するプログラム。決まったものしかできないので使うことないと思います。

Debian Live Project のみで配布 (Debian 非公式パッケージ)

- live-manual: マニュアル
- live-initscripts: 起動時にスクリプトを実行するオプションを追加する
- live-helper, live-initramfs, live-magic のスナップショット

Debian Live の開発はとても早いので、興味があるなら、Debian Live Project で配布されているスナップショット版か git のものを使うとよいでしょう。スナップショットの在り処などは、Debian Live Project のリンクページ[*4]に掲載されています。

live-helper は Debian に依存しないように作られています。Debian 以外のシステムでも以下の要件を満たしていれば、live-helper を使って Debian Live を作成することができます。

- Linux 2.6.x
- POSIX 準拠のシェル (bash や dash など)
- 管理者 (root) 権限
- debootstrap もしくは cdebootstrap
- live-helper

4.11 まとめ

Debian Live を使っていて、日本語のまとまった資料があまりないので頑張って書いてみましたが、いかがでしたでしょうか? 今回は CD/DVD/USB メモリ起動に絞って書きましたが、ネットワーク越しのブートや、live-helper の構造など、まだまだ書ききれない部分が残っているので、それはまた日を改めてということで、皆様のお役に立てればさいわいです。

[*4] DebianLiveProject:http://debian-live.alioth.debian.org/links.html

参考文献

[1] Debian Live Project: http://debian-live.alioth.debian.org/

[2] DebianLive - Debian Wiki: http://wiki.debian.org/DebianLive

[3] DebianLive/FAQ - Debian Wiki: http://wiki.debian.org/DebianLive/FAQ

[4] Debian Live Manual: http://live.debian.net/manual/html/index.html

[5] Daniel Baumann - Blog: Debian Live Initiative
http://blog.daniel-baumann.ch/2006/02/14#20060214_debian-live-initiative

[6] 第 34 回東京エリア Debian 勉強会、2007 年 11 月勉強会:
http://tokyodebian.alioth.debian.org/2007-11.html

[7] Software Design 2008 年 8 月号
live-helper で構築するオリジナル Live CD 岩松信洋

[8] Software Design 2009 年 5 月号
インストール、ネットブック活用から独自バージョンの作成まで - Ubuntu の魅力を体感！
6 章：Inside Ubuntu 吉田史
7 章：カスタム CD イメージの作成 小林準

5 ハッカーに一歩近づく Tips：Bash 編

山下康成@京都府向日市

表 3 Bash コマンド一覧

<<コマンドの再実行>>		
1	以前実行したコマンドラインの呼出し	上矢印キー、下矢印キー
2	以前実行したコマンドラインの呼出し	CTRL-P, CTRL-N
3	直前のコマンドの再実行	!!
4	以前実行したコマンドラインに前方一致	!（前方一致文字列）
5	以前実行したコマンドラインに部分一致	!?（部分一致文字列）
6	これまでに実行したコマンドラインの列挙	history
7	ヒストリ番号を用いた過去コマンド指定	!(数字)
8	指定した回数前に実行したコマンド	!（負の数字）
9	実行されるコマンドの確認	:p
<<補完>>		
10	パス名の補完	TAB
11	コマンドの補完	TAB
<<行編集>>		
12	カーソルの左右移動	CTRL-B/左矢印、CTRL-F/右矢印
13	カーソルの左の文字の消去	BackSpace/Delete
14	カーソルの行頭・行末への移動	CTRL-A, CTRL-E
15	カーソル位置の文字を消す	CTRL-D
16	行末まで消去	CTRL-K
17	語頭まで消去	CTRL-W
18	文字の入れ換え	CTRL-T
<<直前のコマンドの一部を再利用する>>		
19	直前のコマンドの最後の引数	!$
20	直前のコマンドの引数	!*
21	直前のコマンドラインの一部を変更	^A^B
22	直前のコマンドラインの一部を変更	（以前のコマンド指定）:s/A/B/
<<ワイルドカード>>		
23	.（ドット）で始まる以外に一致	*
24	1 文字に一致	?
<<プログラムの実行結果を利用する>>		
25	パイプ	|
26	リダイレクト	> >> <
27	コマンドの実行結果を文字列として扱う	‘（逆シングルクォート）
<<その他>>		
28	ホームディレクトリを指す	~
29	指定したユーザのホームディレクトリを指す	~ユーザ名
30	（コマンドを中断する）	CTRL-C

5.1 参考資料

- ハッカーに一歩近づく Tips: Bash 編
  `http://www.yamasita.jp/tips/bash/`

6 Debian での数学ことはじめ。

まえだこうへい

6.1 はじめに

Debian で数値計算や統計処理、グラフ作成を行うには、様々なパッケージがあります。私のように大学での専行が理系といっても基礎生物学だった人間には、コンピュータでの専用ツールでなくても、極端な話、基本的な統計学と関数電卓があれば事足りました。スプレッドシートを使っても、関数は使わずに電卓代わりに使うことも多かったと記憶してます[*5]。社会人になってからもその延長でスプレッドシートで大抵は事足りる、という方が一般的には多いのではないかと思います。

とはいえ、数万単位のサンプル数の計算をしたり、グラフを描写する場合、スプレッドシートでは非力だという問題もあります。プリセールスや企画で裏付け資料を作るのにサンプルデータを統計処理したり、データをプロットしてグラフを描くと、スプレッドシートだとマウスの操作だけで簡単にできる一方、CPU やメモリ不足でレスポンスが遅くなったり、ソフトウェア自体が落ちたり、酷いと OS 自体のレスポンスも悪化するというのが最近の仕事上での悩みだったりします。

そんなことがきっかけで今頃、Gnuplot を使い始めたわけですが、統計処理では最近は R の書籍を書店で見かけたり、数値計算では Octave というのがあるということを Debian 勉強会の ML で知ったり[*6]と、自分の中だけでなく、最近意外とまわりでもこの辺はホットな話題ではないかと思います。そこで、今回は私と同じように入門者向けに Octave、Gnuplot との連携方法、R の話に加え、Debian での使い方、パッケージングなどについて説明しようと思います。

6.2 概要

GNU Ocatave は計算処理のための高級言語で、MATLAB という数値計算の商用ソフトウェアとの互換性があるようです。行列計算、連立方程式、微分・積分、グラフ出力などに使えます。グラフ描画のためには、gnuplot が必要です。

一方、GNU R は統計解析のためのプログラミング言語で、文法的には、S 言語や Scheme に影響を受けているようです。Octave とは異なり、グラフ描画には gnuplot は必要ないようです。Excel などのデータ読み込みができるといった、データの互換性にも特徴があります。

[*5] 理由は、当時のスプレッドシートの関数の精度が低く、自分で計算式を作った方が確実だったためです。今では改善されたのでしょうか？

[*6] R や Octave のパッケージメンテナである Dirk Eddelbuettel さんが近々来日されるというのが ML で流れていたそもそものきっかけです。`http://dirk.eddelbuettel.com/`

6.3 インストール方法

Debian での各実行環境の導入方法について見てみます。今回の前提条件として、ディストリビューションは Sid[*7] としていますが、Stable(Lenny) や Testing(Squeeze) でもバージョンが異なることを除けばほぼ同じです。適宜読み替えてください。

6.3.1 Octave のインストール方法

GNU Octave は Squeeze/Sid では、`octave` というパッケージ名ではなく、バージョン 3.0 と 3.2 で別のパッケージになっています。今回は 3.2 のパッケージを導入します。

```
$ sudo apt-get install octave3.2
```

6.3.2 GNU R のインストール方法

GNU R は、Debian パッケージでの名前は、gnur でも、r でも gnu-r でもありません。`r-base` というメタパッケージで提供されています。これをインストールすると、最低限、`r-base-core` パッケージが GNU R の実行には必要です。また、GNU R には、機能拡張のパッケージが用意されています。CRAN(the Comprehensive R Archive Network) からダウンロードできます。Perl での CPAN のような位置づけです。千以上ものパッケージがあるようです。Debian では、CPAN や Ruby gems と同様にここから直接ダウンロードすることもできますが、約 150 の R パッケージが deb パッケージとして用意されています[*8]。パッケージ名は"`r-cran-<name>`" というネーミングルールです。

今回は `r-base` をインストールしましたが、余計なパッケージを入れないようにと、`r-base-core` パッケージをインストールしたとしても最小構成からのパッケージの差は二つしかありません。[*9]

```
$ sudo apt-get install r-base
```

6.3.3 Gnuplot のインストール

Gnuplot は Octave、GNU R のいずれも deb パッケージとしての依存関係はありませんが、octave では gnuplot を使ってグラフ描画もできるので、インストールしておくと便利でしょう。gnuplot のインストールには `gnuplot` パッケージをインストールします。

```
$ sudo apt-get install gnuplot
```

6.4 試してみる。

今回は、我が家の過去 9 年分の光熱費における各使用量の分布をサンプルデータとしてみます。まずは、もともと使い始めていた gnuplot での例をみて、それに対し、Octave, R ではどのように処理するのかを見てみましょう。

用意したサンプルデータは以下のようなものです。

```
"date","days","kWh","kWh/d","yen/d","yen","days","m^3","m^3/d","yen/d","yen","days","m^3","m^3/d","yen/d","yen","yen/d","yen"
2001/04,11,34,3.09,83.5,918,36,4.9,0.14,108.3,3900,58,9,0.16,22.8,1320,,
2001/05,33,95,2.88,73.8,2434,29,3.5,0.12,114.1,3310,,,,,,,
2001/06,30,129,4.3,102.7,3082,32,3,0.09,96.9,3100,61,3,0.05,15.7,960,,
2001/07,28,155,5.54,131.3,3676,28,1.7,0.06,91.1,2550,,,,,,,
(snip)
```

左から、年月, 電気使用日数, 月の電気使用量 (kWh),1 日あたりの電気使用量,1 日あたりの電気代, 月の電気代, ガ

[*7] 2009 年 11 月 10 日現在。
[*8] 2009 年 11 月 9 日現在。
[*9] 一つは `r-base`, もう一つは `r-base-dev` です。

ス使用日数, 月のガス使用量 (立方メートル),1 日あたりのガス使用量,1 日あたりのガス代, 月のガス代, 水道使用日数, 水道使用量 (立方メートル),1 日あたりの水道使用量,1 日あたりの水道代, 月の水道代,1 日あたりの下水道代, 月の下水道代、となっています。

6.4.1 gnuplot の場合

gnuplot では、次のようなバッチファイルを用意します。2001 年 4 月から 2009 年 8 月までの電気、ガス、水道の各使用量をそれぞれプロットします。また、Debian 勉強会資料用に、LaTeX で取り込めるように、図は EPS、文字列は LaTeX で出力するようにします。

```
reset
set terminal epslatex input color
set output "gnuplot.tex"
set datafile separator ","
set xdata time
set timefmt "%Y/%m"
set format x "%Y/%m"
set xtics rotate by -90
set mxtics 12
set xrange ["2001/04":"2009/08"]
set yrange [0:400]
set xlabel "年月"
set ylabel "電気使用量 [kWh]"
set y2range [0:50]
set y2label "ガス・水道使用量［立方メートル］"
set ytics nomirror
set y2tics
plot "kohnetsuhi.csv" using 1:3 axes x1y1 title "電気" with line,\
     "" using 1:8 axes x1y2 title "ガス" with line,\
     "" using 1:13 axes x1y2 title "水道" with line
```

このバッチファイルを LaTeX と EPS に変換します。バッチファイルをロードするには、対話モードで load コマンドを実行します。

```
$ gnuplot

        G N U P L O T
        Version 4.2 patchlevel 6
        last modified Sep 2009
        System: Linux 2.6.31.3

        Copyright (C) 1986 - 1993, 1998, 2004, 2007 - 2009
        Thomas Williams, Colin Kelley and many others

        Type ‘help‘ to access the on-line reference manual.
        The gnuplot FAQ is available from http://www.gnuplot.info/faq/

        Send bug reports and suggestions to <http://sourceforge.net/projects/gnuplot>


Terminal type set to ’wxt’
gnuplot> load ‘‘gnuplot.gp’’
gnuplot> quit
```

これで、カレントディレクトリに gnuplot.tex と gnuplot.eps ファイルが出力されます。gnuplot.tex は、LaTeX 文書の中で以下のように記述して取り込みます。

```
\usepackage{graphicx}

(snip)

\begin{figure}[thbp]
\input{image200911/gnuplot.tex}
\end{figure}
```

ただし、このままでは make が通りません。gnuplot.tex を書き換える必要があります。次のように下から 3 行目の includegraphics のパスを変更する必要があります。

```
$ diff -u gnuplot.tex.bk gnuplot.tex
--- gnuplot.tex.orig        2009-11-12 23:38:36.000000000 +0900
+++ gnuplot.tex         2009-11-12 23:39:39.000000000 +0900
@@ -111,7 +111,7 @@
       \put(5003,4163){\makebox(0,0)[r]{\strut{}水道}}%
     }%
     \gplbacktext
-    \put(0,0){\includegraphics{gnuplot}}%
+    \put(0,0){\includegraphics{image200911/gnuplot.eps}}%
     \gplfronttext
   \end{picture}%
 \endgroup
\end{document}
```

これで make すると、以下のような図を表示できます。

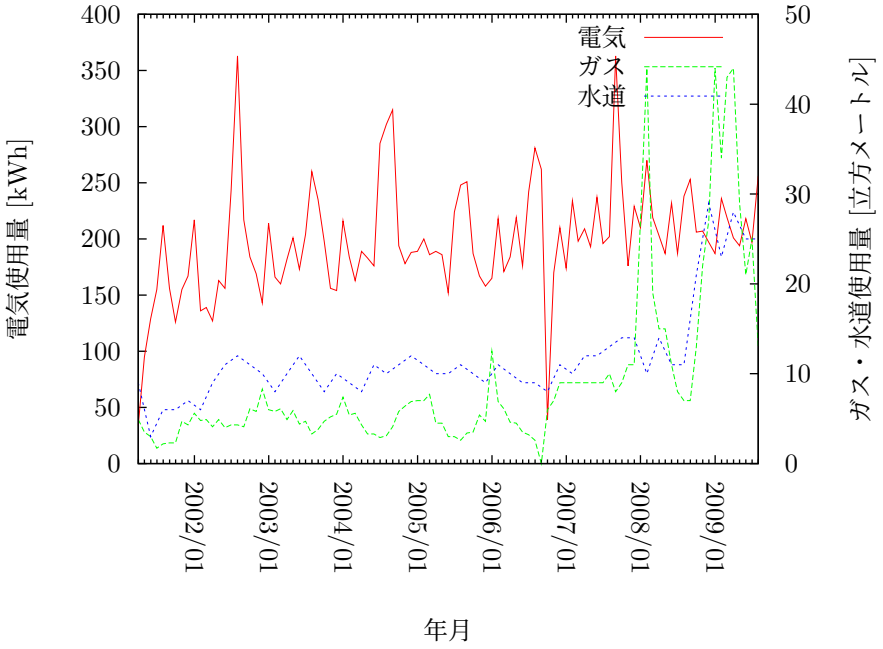


図 1　gnuplot でのプロットの例

では、これを R ではどのようにするのかを見てみましょう。[*10]

6.4.2　GNU R の場合

GNU R を対話形式で使うには、シェルで R と実行します。

```
$ R
```

R でデータを扱うには、先ほどの konetsu.csv ファイルをオブジェクトに取り込む必要があります。取り込むには、<-演算子を使います。

```
> konetsu <- read.csv("konetsu.csv")
```

この konetsu オブジェクトの中身を見てみます。このオブジェクトはデータフレームといい、これを指定するだけです。

[*10] Octave は gnuplot を使ってグラフ描画するということなので、今回省略。手が回らなかったのが事実ですが、何か。

```
> konetsu
    X...date days kWh kWh.day yen.day  yen days.1  m.3 m.3.day yen.day.1 yen.1
1     2001/4   11  34    3.09    83.5  918     36  4.9    0.14     108.3  3900
2     2001/5   33  95    2.88    73.8 2434     29  3.5    0.12     114.1  3310
3     2001/6   30 129    4.30   102.7 3082     32  3.0    0.09      96.9  3100
4     2001/7   28 155    5.54   131.3 3676     28  1.7    0.06      91.1  2550
5     2001/8   34 212    6.24   146.9 4995     35  2.2    0.06      78.9  2760
(snip)
    days.2 m.3.1 m.3.day.1 fee.d yen.2 yen.day.2 yen.3
1       58     9      0.16  22.8  1320        NA    NA
2       NA    NA        NA    NA    NA        NA    NA
3       61     3      0.05  15.7   960        NA    NA
4       NA    NA        NA    NA    NA        NA    NA
5       60     6      0.10  19.0  1140        NA    NA
(snip)
```

列数が増えると、この例のように分割して表示されます。“NA” と表示されているのは、値が null だったものです。

また、この取り込んだデータのサマリを表示させることも簡単です。以下のように R では統計処理のための関数があらかじめ用意されているので、とても便利です。

```
> summary(konetsu)
    X...date       days            kWh           kWh.day          yen.day
 2006/9 : 2   Min.   :11.00   Min.   : 34.0   Min.   : 2.880   Min.   : 73.8
 2001/10: 1   1st Qu.:29.00   1st Qu.:170.2   1st Qu.: 5.753   1st Qu.:129.1
 2001/11: 1   Median :30.00   Median :195.0   Median : 6.410   Median :147.3
 2001/12: 1   Mean   :30.09   Mean   :199.1   Mean   : 6.565   Mean   :151.3
 2001/4 : 1   3rd Qu.:33.00   3rd Qu.:219.0   3rd Qu.: 7.225   3rd Qu.:171.0
 2001/5 : 1   Max.   :35.00   Max.   :363.0   Max.   :11.000   Max.   :250.1
 (Other):95
(snip)
```

各項目のデータを表示するには、データフレームのオブジェクト名と自動的につけられた変量名を **データフレーム名$変量名** で指定します。

```
> konetsu$date
  [1] 2001/04 2001/05 2001/06 2001/07 2001/08 2001/09 2001/10 2001/11 2001/12
 [10] 2002/01 2002/02 2002/03 2002/04 2002/05 2002/06 2002/07 2002/08 2002/09
(snip)
[100] 2009/07 2009/08 2009/09
102 Levels: 2001/04 2001/05 2001/06 2001/07 2001/08 2001/09 2001/10 ... 2009/09
```

それでは、gnuplot の時と同じようにプロットしてみます。今回も EPS で出力します。

```
> postscript("gnur.eps", horizontal=FALSE, height=5, width=5, pointsize=10)
> plot(konetsu$kWh, type="l", ylim=c(0,400), ann=F)
> par(new=T)
> plot(konetsu$m.3, type="l", ylim=c(0,400), ann=F, col="red")
> par(new=T)
> plot(konetsu$m.3.1, ylim=c(0,400), ann=F, col="blue")
> dev.off()
X11cairo
       2
>
```

出力された、EPS を LaTeX に取り込むと以下の図のようになります。

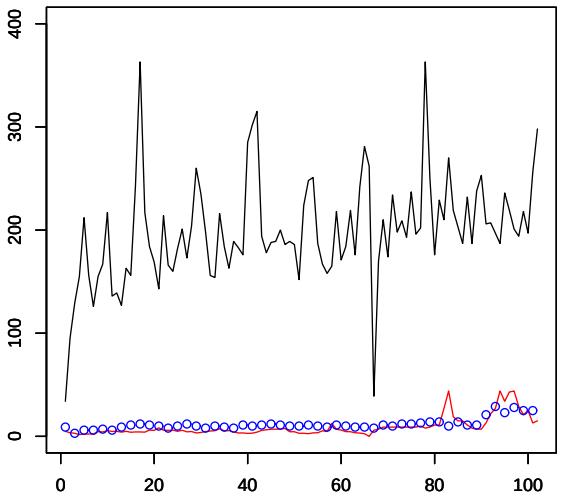


図 2　GNU R でのプロットの例

第 2 軸だけに表示させる方法が分からなかったので、Y 軸のスケールを全部同じにしなくてはならなかったりと、gnuplot に比べると若干見づらくなってしまいました。もうちょっと修行が必要そうです。

6.5　Debian の環境における注意点。

ところで、Debian では GNU R は CRAN のパッケージが deb パッケージ化されていると上述しましたが、GNU Octave も Octave-Forge プロジェクトという、CRAN に相当する拡張パッケージが、SorceForge で公開されています[*11]。CRAN 由来のパッケージと違い、パッケージのネーミングルールは決まっていません。ただし、現状では、Octave そのものに由来するパッケージは、`octaveX.X[-*]` という形で、バージョン X.X のブランチごとのパッケージ、名前になっています。一方、Octave-Forge をはじめ、拡張機能のパッケージは、`octave-*`というパッケージ名になっていますので、ある程度判断するための目安にはなるでしょう。

```
$ apt-cache search "GNU R" | wc -l
213
$ apt-cache search octave | wc -l
133
```

それぞれ、deb パッケージにする場合の考慮点を見てみましょう。

6.5.1　Octave の場合

Debian には Octave 関連パッケージのメンテナンスチーム Debian Octave Group が存在します[*12]。deb パッケージに未だなっておらず、deb パッケージにしたいソフトウェア、例えば、膨大な CRAN パッケージを deb パッケージ化したい場合、あるいは野良パッケージを作った場合などは、ITP する際に、Debian Octave Group[*13]にも CC を入れておくと良いでしょう。

また、README.Debian にも記載されていますが、Octave のドキュメントには PDF が含まれているため、`octave3.0-doc, octave3.2-doc` という名前で拡張パッケージとして提供されています。

Octave パッケージのビルドには、`octave-pkg-dev` パッケージという専用の環境が用意されています。Octave は

[*11] `http://octave.sourceforge.net/packages.html`
[*12] `http://pkg-octave.alioth.debian.org`
[*13] pkg-octave-devel@lists.alioth.debian.org

バージョン 3.0 からアドオンを作成する場合には、pkg.m system という環境を使ってインストールするようになっていますが、これを Debian 用のパッケージング環境として用意されています。

このパッケージを使ってパッケージを作るには、debian/control の Build-Depends に、octave-pkg-dev を、Depends に`${octave:Depends}`を追記します。

また、debian/rules に

```
include /usr/share/cdbs/1/class/octave-pkg.mk
```

を追記する必要があります[*14]。

また、debian/rules で、get-orig-source ターゲットも透過的に提供されています。これは前述の Octave-Forge プロジェクトからのパッケージの配布のみに作用しています。アップストリームのパッケージがこの SourceForge 以外からする場合は、自分で get-orig-source ターゲットを定義できます。octave-pkg-dev パッケージ特有のルールの実行を防ぐには、`''SOURCEFORGE = NO''` を debian/rules に設定する必要があります。

6.5.2 GNU R の場合

GNU R パッケージのビルドには、Octave と同様、`r-base-dev` パッケージという専用のビルド環境が用意されています。ビルド時の対応として、Debian パッケージポリシーの考慮が必要です。R の実行ファイルは、/usr/lib/R/bin/R.binary として分離されています。

私の環境では、上記のパスには以下のファイルが存在しますが、ここでは、/usr/lib/R/bin/Rscript が該当します。

```
$ file /usr/lib/R/bin/R*
/usr/lib/R/bin/R:       Bourne-Again shell script text executable
/usr/lib/R/bin/REMOVE:  ASCII text
/usr/lib/R/bin/Rcmd:    Bourne-Again shell script text executable
/usr/lib/R/bin/Rd2dvi:  ASCII English text
/usr/lib/R/bin/Rdconv:  ASCII text
/usr/lib/R/bin/Rdiff:   Bourne-Again shell script text executable
/usr/lib/R/bin/Rprof:   a /usr/bin/perl script text executable
/usr/lib/R/bin/Rscript: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
 dynamically linked (uses shared libs), for GNU/Linux 2.6.18, stripped
```

6.6 まとめ

今回は、Debian で数学ことはじめ、ということで、Octave, GNU R, gnuplot を取り上げ、基本的な使い方と、ちょっと視点を変えて、拡張パッケージの Debian パッケージのお作法についてお話しました。使い込んでいけばかなり便利そうな印象があります。

数学、と言う観点からはかなりずれてしまいましたが、スプレッドシート地獄から脱出しましょう。

参考文献

[1] 山本昌志「gnuplot の精義 フリーの高機能グラフ作成ツールを使いこなす」

[*14] octave-pkg-dev パッケージの README には、include /usr/share/octave/debian/octave-pkg-dev.mk を追記するように記述されていますが、バージョン 0.5 でファイル名の変更をしており、ドキュメントの更新が追いついていないようです。

7 デバッグのお供："gdb のススメ"

杉本典充

7.1 はじめに

インターネットでは様々なオープンソースのプログラムが公開されています。それらのプログラムは開発者の手によるデバッグだけでなく、多くの人もデバッグ作業に参画することによって品質を高めていきます。そのデバッグ作業を終えたプログラムがいわゆる「安定したプログラム」であり、多くの人が安心して使えるレベルになるにはデバッグ作業はとても大切な作業の 1 つです。今回は、Debian を使ってプログラムをデバッグする手法についてまとめてみました。

7.2 gdb とは

gdb とは GNU Debugger[*15]のことで、C 言語・C++ 向けのソースレベルデバッガです。開発者は gdb を使うことでプログラムが今どこの部分を実行しているか、プログラムの状態はどうなっているかを知ることができるため、デバッグ作業を効率的に行うことができます。「man gdb(1)」によると、gdb には大きく 4 つの機能があると書かれています。

- プログラムの動作を詳細に指定してプログラムを実行させる。
- 指定した条件でプログラムを停止させる。
- プログラムが止まった時に、何が起こったか調べる。
- バグによる副作用を修正し、別のバグを調べるためプログラムの状態を変更する。

gdb が使用する設定ファイルは"～/.gdbinit"であり、gdb の初期設定値をこのファイルに定義することで変更できます。

7.3 開発環境と gdb のインストール

C 言語のプログラムを開発するためにはコンパイラが必要です。gdb の他にプログラムを作成するために必要なソフトウェア一式もインストールします。

```
$ sudo apt-get update
$ sudo apt-get install gcc make
$ sudo apt-get install gdb
```

環境も整ったところで、プログラムを作成します。今回は「FizzBuzz」[*16]といわれているプログラムを例にしてみ

[*15] GNU Debugger Web サイト `http://www.gnu.org/software/gdb/`

[*16] FizzBuzz とは、『1 から 100 からまでの整数を標準出力に出力せよ。ただし、3 で割り切れるときは「Fizz」、5 で割り切れるときは「Buzz」、3 と 5 の両方で割り切れるときは「FizzBuzz」と標準出力に出力せよ。』というプログラムの問題である。

ます。

7.4 gdb を使ってみましょう

7.4.1 まずはデバッグビルドします

プログラムを gdb で操作するためにはプログラムにデバッグ情報を付与してビルドする必要があります。gcc のコンパイルオプション及びビルドオプションにデバッグシンボルを付与する”-g”オプションをつけてビルドします。(デバッグ時の最適化レベルは開発者によって指定が違うこともあります。ここでは最適化レベルは無指定 (=”-O0”、最適化なし) としてビルドします。)

7.4.2 gdb 単体でプログラムを追いかけてみる

それではシェルから gdb を起動します。gdb を起動すると以下のような入力受付状態になります。

```
$ gdb
GNU gdb 6.8-debian
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
(gdb)
```

gdb のプロンプトでコマンドを入力することによりデバッガを通してプログラムを動かすことができます。デバッグ作業でよく使う gdb のコマンドを表 7.4 に示します。

7.5 gdb のフロントエンドツール

gdb は単体でも十分デバッグ可能ですが、よりデバッグ作業を行いやすいように gdb のフロントエンドツールが多くあります。X Window System 上で動作する統合開発環境 (IDE) では KDevelop、Anjuta、Eclipse、NetBeans など、コマンドライン上でも動作する Emacs、Vim などもフロントエンドとして利用することができます。

7.5.1 Emacs GUD モードで gdb を使ってみる

Emacs には GUD(Grand Unified Debugger) という機能があり、Emacs 上で様々なデバッガと連携することができる仕組みです。GUD は gdb に限らず、perldb(perl 用デバッガ) や pdb(python 用デバッガ) なども起動することができます。

Emacs の実行中に以下のキーを入力して gdb を起動します。

```
M-x gdb
```

その後、ミニバッファで実行ファイルを指定して Enter キーを入力します。

```
Run gdb (like this): gdb --annotate=3 ../a.out
```

すると図 7.3、図 7.4 のような画面に切り替わります。

表 7.4　gdb を操作するコマンド (一部抜粋)

コマンド	省略コマンド	説明
run (引数)	r	プログラムを最初から実行します。 引数を指定する場合は、run の後に引数を指定します。
break (停止位置)	b	ブレークポイントを設定します。ブレークポイントは「関数名」と「ソースコード:行番号」のいずれかで指定できます。
delete (breakpoint 番号)	d	ブレークポイントを削除します。単に delete だけを実行するとすべてのブレークポイントを削除します。
list	l	現在実行中の近くのソースコードをある行数表示します。(初期設定値は 10 行)
step	s	ステップイン実行します。
next	n	ステップオーバー実行します。
finish	fin	ステップアウト実行します。
continue	c	現在停止中の位置からプログラムを再開します。
print (変数名)	p	プログラム中の (変数名) の内容を表示します。ポインタ変数の場合は「print *ポインタ変数名」と指定することでポインタが指し示す値を表示できます。
set var (変数名)=(設定値)	なし	プログラム中の (変数名) の値を (設定値) に変更します。
quit	q	gdb を終了します。
attach (プロセス ID)	なし	実行中の (プロセス ID) を gdb で制御できるようにします。
detach	なし	attach 中のプロセスを gdb の制御下から切り離します。切り離されたプログラムはそのまま動作し続けます。
shell	なし	shell を起動します。shell を exit すると gdb プロンプトに戻ります。
help	h	gdb のコマンドに関するヘルプを表示します。
info (コマンド)	i	様々な情報を表示します。

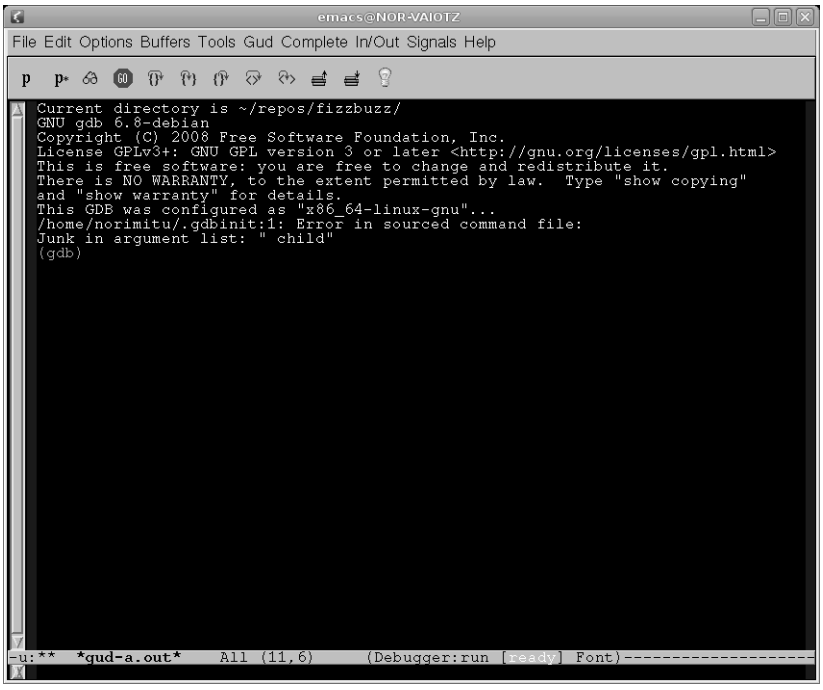


図 7.3　Emacs GUD モードで gdb を起動した画面 (1)

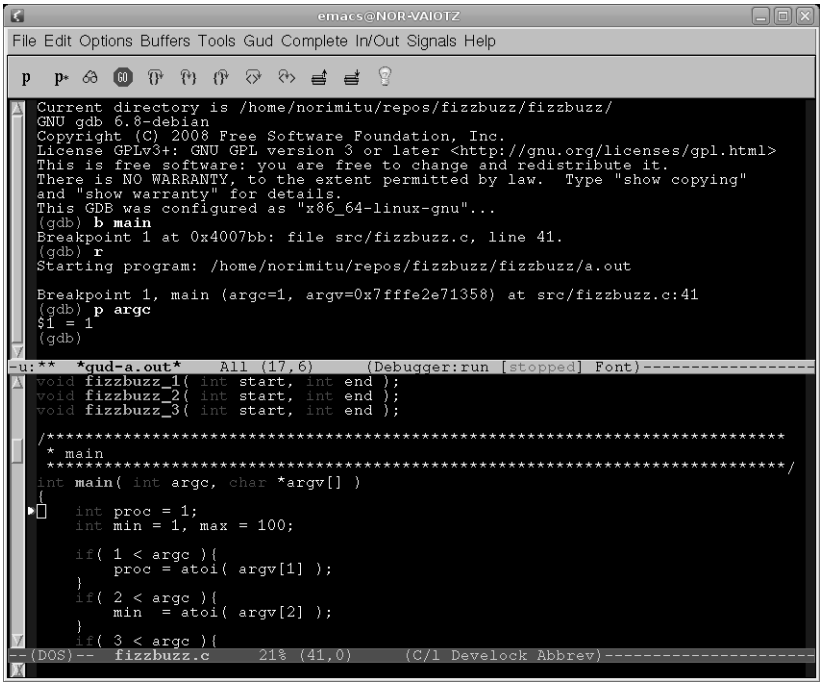


図 7.4　Emacs GUD モードでデバッグ中の画面 (2)

また、Emacs の設定ファイル”〜/.emacs.el”に「(setq gdb-many-windows t)」を指定しておくと、すると図 7.5、図 7.6 のような画面で gdb が起動します。この画面を表示するには「gud.el」というファイルが必要であり。lenny の場合は Emacs をインストールすると一緒にインストールされます。

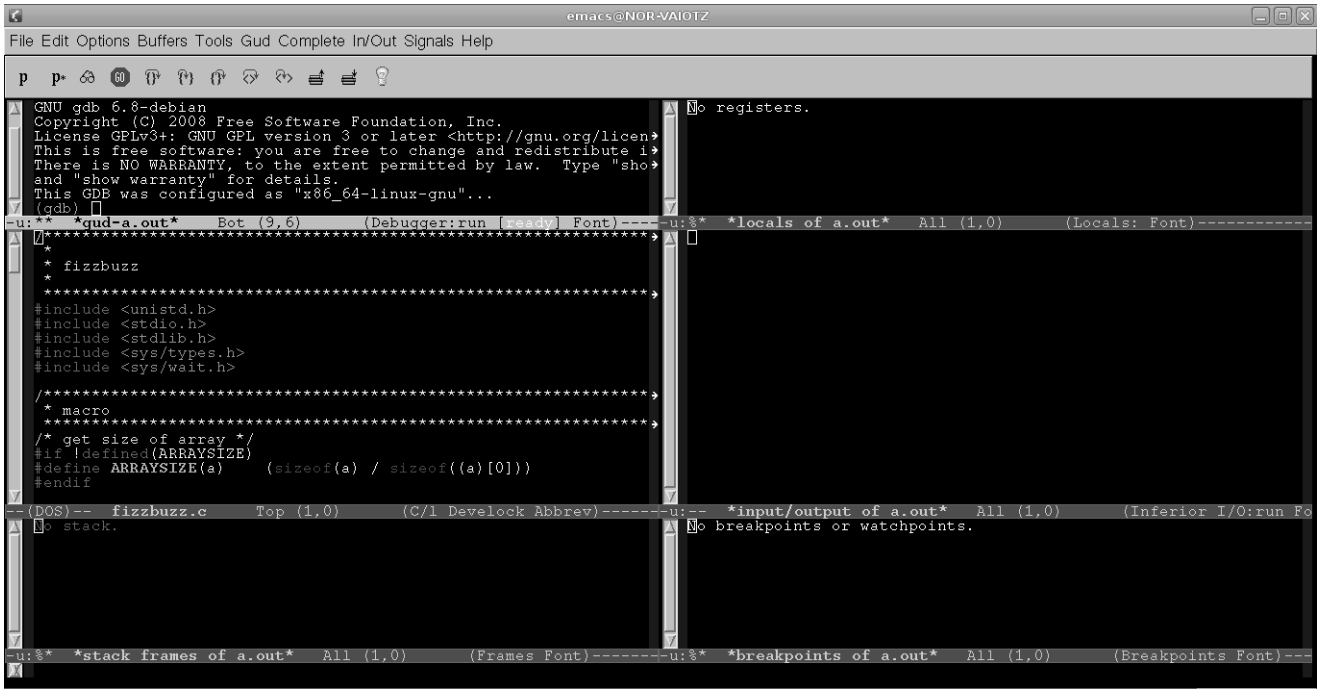


図 7.5　Emacs GUD モードで gdb を起動した画面 (3)

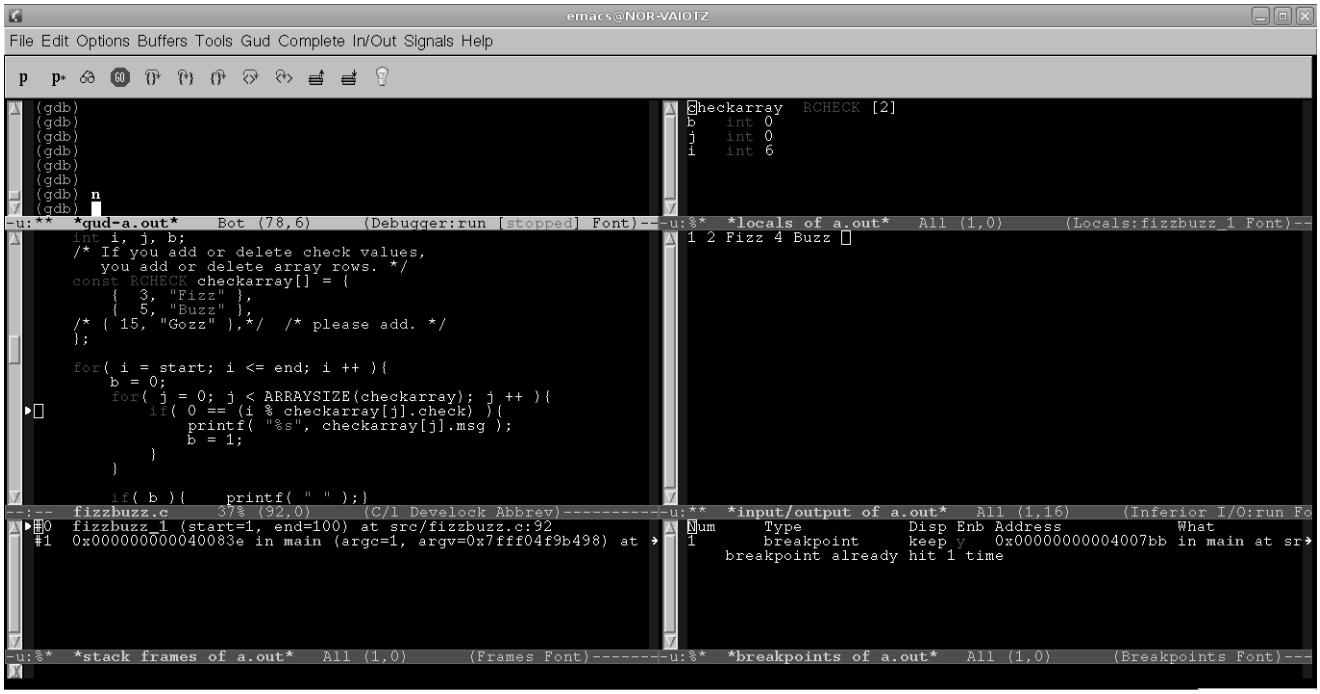


図 7.6　Emacs GUD モードでデバッグ中の画面 (4)

7.6　いろいろなプログラムのデバッグ方法

私がプログラムを gdb を使ってデバッグするときの操作例を挙げてみます。

7.6.1　単発実行系プログラムのデバッグ

単発実行するプログラムの場合はデバッガでプログラムの起動を行い、その後にデバッグ作業を開始することになります。

1. gdb を起動します。
2. break コマンドでブレークポイントを指定します。(実際には「b main」と入力してプログラムの最初で止めることも多いです。)
3. run コマンドでプログラムを開始します。
4. step コマンド、next コマンドでプログラムを追いかけます。
5. デバッグが完了したら、continue コマンドで残りのプログラムすべてを実行します。
6. quit コマンドで gdb を終了します。

7.6.2　デーモン系プログラムのデバッグ

デーモンとして動作しているプログラムの場合は、動作中のプログラムを gdb で制御する必要があるためアタッチする必要があります。

1. デバッグするデーモンプログラムを実行します。
2. デバッグするデーモンプロセスのプロセス ID を調べます。
3. gdb を起動します。
4. プロセス ID を指定して attach コマンドを実行し、デバッグするデーモンプロセスにアタッチします。

5. break コマンドでブレークポイントを指定します。(おそらく無限ループ処理のどこかで停止させることになると思います。)
6. ブレークポイントを設定したところでプログラムが一時停止しますので、step コマンドや next コマンドを実行してプログラムを追いかけます。
7. デバッグが終了したら、detach コマンドでプロセスからデタッチします。
8. quit コマンドで gdb を終了します。

7.6.3 fork するプログラムのデバッグ

fork するプログラムの場合、fork() 後に親プロセスと子プロセスのどちらを追いかけるのかを「set follow-fork-mode parent」などと設定しておく必要があります。

1. gdb を起動します。
2.「set follow-fork-mode」を設定し、fork() 後にデバッガを追うプロセスを親プロセスにするか、子プロセスにするか設定します。
3. break コマンドでブレークポイントを指定します。
4. run コマンドでプログラムを開始します。
5. fork() した後は「set follow-fork-mode」で指定した親プロセスか子プロセスのいずれかを追従しますのでそのまま続けてデバッグします。
6. quit コマンドでプログラムを終了します。

7.7 まとめ

今回は gdb の紹介と Emacs GUD モードにおいてプログラムをデバッグする一例を紹介しました。Emacs GUD モードを使ったデバッグ操作は X Window System 上だけでなくコンソール環境でも同様の手順で実行できるため、telnet 環境や ssh 環境でも同じスタイルでプログラムのデバッグ作業を行うことができます。

みなさんも Debian を使ってたくさんデバッグしてみましょう。

7.8 参考資料

- man gdb(1)

8 GPG キーサインパーティの説明

岩松 信洋

8.1 なぜキーサインするのか？

- PGP/GnuPG は認証局がないので、自分が相手を信頼するしかありません。
- 相手を信頼するには、キーサインパーティを行って PGP/GnuPG の公開鍵をソーシャルな情報とともに交換し、信頼の輪 (web of trust) を広げる必要があります。

8.2 使いどころ：フリーソフトウェア開発者の場合

- アカウント作成時のチェックに利用する場合があります。（インターネット上での存在を示す。）
- ソフトウェアのリリース時に利用する場合があります。
- Debian ではパッケージへの署名、投票などに使います。

8.3 使いどころ：ユーザの場合は？

- メールへの署名／暗号化に利用する場合があります。
- Debian Project 公式開発者になるための通過儀礼の一つです。

身近なところでは、改竄のチェックに使います。Debian の場合は secure-apt で使われています。

Debian アーカイブの鍵が更新されていなかったり、アップデートされていないと以下のようになります。

```
# apt-get update
.....
W: GPG error: http://cdn.debian.or.jp testing Release: The following
 signatures couldn't be verified because the public key is not
 available: NO_PUBKEY 9AA38DCD55BE302B
```

このような状態になった場合には、GPG の鍵サーバから secure-apt 用の最新の鍵を取得し、更新します。最新の鍵については `http://ftp-master.debian.org/keys.html` を参照してください。以下に更新の例を示します。

```
# gpg --keyserver wwwkeys.eu.pgp.net --recv-keys 9AA38DCD55BE302B
# gpg --armor --export 9AA38DCD55BE302B | apt-key add -
# apt-get update
.....
Fetched 2B in 1s (1B/s)
Reading package lists... Done
```

何も考えずにやると上記のようになりますが、これではいけません。鍵を更新する前にちゃんと鍵と信頼度をチェックしましょう。**エラーがでなくなった！これで大丈夫！(って書いてある Web サイト多いよね。) と思ってはだめです。**もちろん、鍵のチェックには Web Of Trust に入らないとできません。

8.4 チェックする簡単な方法

では、鍵をチェックするにはどうしたらいいのでしょうか。例えば、9AA38DCD55BE302B の鍵に署名している人は以下のとおりです。

```
pub  4096R/55BE302B 2009-01-27
uid Debian Archive Automatic Signing Key (5.0/lenny) <ftpmaster@debian.org>
sig  sig3  55BE302B 2009-01-27 _____ 2012-12-31 [selfsig]
sig  sig   7E7B8AC9 2009-01-27 _____ __________ Joerg Jaspert <joerg@debian.org>
sig  sig   D0EC0723 2009-01-27 _____ __________ Mark Hymers <mhy@debian.org>
sig  sig   BE9BF8DA 2009-01-27 _____ __________ Mike O'Connor (stew) <stew@vireo.org>
sig  sig   30B94B5C 2009-05-24 _____ __________ XXXXXXXXX (imacat) <imacat@mail.imacat.idv.tw>
```

*17

まず、公開鍵を取得し鍵のチェックを行います。

```
$ gpg --keyserver pgp.mit.edu --recv-keys 55BE302B
$ gpg --list-sig 55BE302B
pub   4096R/55BE302B 2009-01-27 [満了: 2012-12-31]
uid                  Debian Archive Automatic Signing Key (5.0/lenny)
<ftpmaster@debian.org>
sig          7E7B8AC9 2009-01-27  [ユーザー ID が見つかりません]
sig          D0EC0723 2009-01-27  [ユーザー ID が見つかりません]
sig          BE9BF8DA 2009-01-27  [ユーザー ID が見つかりません]
sig          30B94B5C 2009-05-24  [ユーザー ID が見つかりません]
sig 3        55BE302B 2009-01-27  Debian Archive Automatic Signing Key (5.0/lenny) <ftpmaster@debian.org>
```

この結果から、この鍵にサインしている人とは、直接 GPG サインしていないことがわかります。

8.5 trust path finder

しかし Web of Trust なので、信頼のパスが使えます。
trust path finder*18を使うと GPG の信頼のパスが分かります (図 8.1)。

8.6 Joerg と岩松の信頼のパス (trust path)

Debian FTP master である Joerg と岩松の信頼のパスを調べると図 8.2 のようになります。

他の人を介して、Web of Trust がつながっていることが分かります。知り合いの知り合いがサインしているようです。ちょっとは信用できるかな？

8.7 キーサインの流れ

さて、実際のキーサインの流れを見てみましょう。簡単な流れは図 8.3 のようになります。

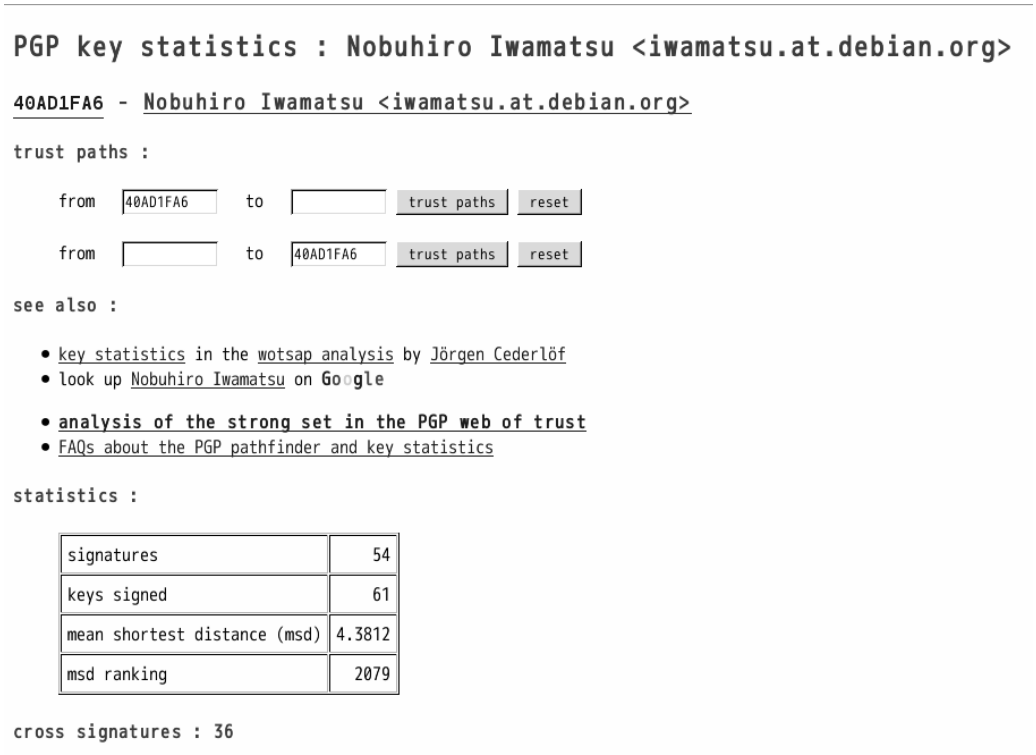
8.8 gpg コマンドでやった場合

キーサインをコマンドで行うと以下のようになります。

```
$ gpg --keyserver pgp.mit.edu --recv-key 40AD1FA6
$ gpg --fingerprint 40AD1FA6
$ gpg --edit-key 40AD1FA6
$ gpg --sign-key 40AD1FA6
$ gpg --check-sig 40AD1FA6
$ gpg --export -a 40AD1FA6 > iwamatsu.gpgkey
$ iwamatsu.gpgkey を 相手にメールに 署名 + 暗号化して送信
```

数人ならいいけど、キーサインパーティでは 100 人ぐらい集まることもあります。100 人分もコマンドであるとな

*17 imacat の名前は文字化けのため XXXXXXXX にしてあります。
*18 http://pgp.cs.uu.nl/mk_path.cgi



PGP key statistics : Nobuhiro Iwamatsu <iwamatsu.at.debian.org>

40AD1FA6 - Nobuhiro Iwamatsu <iwamatsu.at.debian.org>

trust paths :

from 40AD1FA6 to [ ] trust paths reset

from [ ] to 40AD1FA6 trust paths reset

see also :

- key statistics in the wotsap analysis by Jörgen Cederlöf
- look up Nobuhiro Iwamatsu on Google
- analysis of the strong set in the PGP web of trust
- FAQs about the PGP pathfinder and key statistics

statistics :

signatures	54
keys signed	61
mean shortest distance (msd)	4.3812
msd ranking	2079

cross signatures : 36

図 8.1　GPG の信頼のパス

ると大変です。

8.9　そこで caff の登場

上記のような場合、caff を使って行うと作業が大変楽になります。caff は signing-party パッケージで提供されています。インストール・初期化の手順を以下に示します。

インストールはいつもの apt-get で行えます。

```
$ sudo apt-get install signing-party
```

次に caff を使うための初期化を行います。caff を一回実行すると、ホームディレクトリ下に初期ファイル**.caffrc**を作成してくれます。

```
$ caff
.......
#
#Regards,
#{$owner}
#EOM

Please edit /home/hoge/.caffrc and run caff again.
```

8.10　caff の設定

次に ~/.caffrc にある 設定ファイルを修正します。

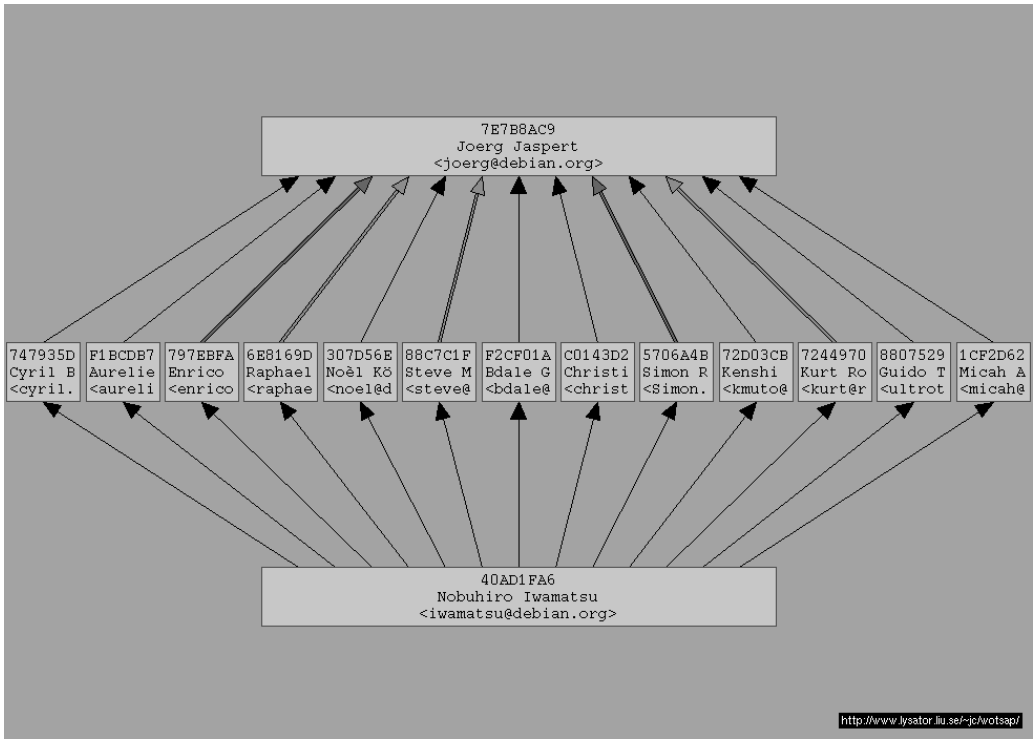


図 8.2　GPG の信頼のパス

```
$ cat ~/.caffrc

$CONFIG{'owner'} = 'Nobuhiro Iwamatsu';
$CONFIG{'email'} = 'iwamatsu@debian.org';

$CONFIG{'keyid'} = [ qw{4121C7433170EBE9 32247FBB40AD1FA6} ]

# Additionally encrypt messages for these keyids
$CONFIG{'also-encrypt-to'} = [ qw{4121C7433170EBE9 32247FBB40AD1FA6} ]

# Mail template to use for the encrypted part
$CONFIG{'mail-template'} = << 'EOM'\maketitle#Hi,

please find attached the user id{(scalar @uids >= 2 ? 's' : '')}
{foreach $uid (@uids) {
    $OUT .= ``\t''.$uid.''\n'';
};}of your key {$key} signed by me.
......
```

8.11　caff の設定

caff のデフォルトの設定では、cert-digest-algo が SHA1 になっているので、[*19] SHA512 に設定します。

```
$ mkdir -p ~/.caff/gnupghome
$ chmod 700 ~/.caff/gnupghome
$ cat >> ~/.caff/gnupghome/gpg.conf
cert-digest-algo SHA512
personal-digest-preferences SHA512
EOF
```

また、サインした鍵をメールで送信するので、ローカル（作業するマシン）の SMTP も設定しておく必要があります。

[*19] http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=527944

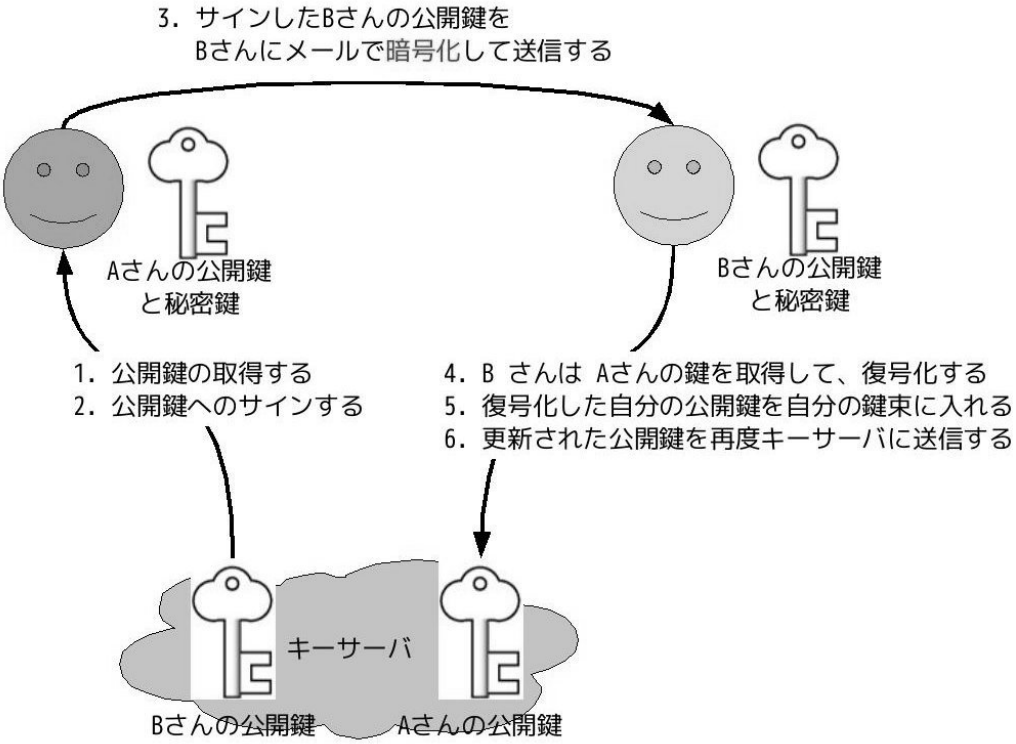


図 8.3　キーサインの流れ

8.12　caff を使った署名

以下のように実行すると、署名する ID をキーサーバから取得し、指定した自分の ID で署名してくれます。そして、署名した鍵を暗号化して送信してくれます。

```
$ caff -u 自分の ID 署名する ID .......
```

8.13　署名完了後

署名後のデータは ~/.gnupg/pubring.gpg ではなく、**~/.caff/gnupghome/pubring.gpg** に格納されています。この鍵束を ~/gnupg/pubring.gpg に 取り込む必要があります。以下の方法で取り込む事が可能です。

```
$ gpg --import ~/.caff/gnupghome/pubring.gpg
```

取り込んだら、自分の鍵をキーサーバに送信します。

```
$ gpg --keyserver pgp.nic.ad.jp --send-keys 自分の ID
$ gpg --keyserver pgp.mit.edu --send-keys 自分の ID
```

8.14　最後に

相手に鍵を送るまでがキーサインパーティです。ちゃんと相手に署名した鍵を送りましょう。

9 DDTSS を使ってみよう

倉敷 悟

9.1 そもそも DDTSS って何？

シンプルに言うと、「パッケージ説明文」を翻訳するための Web インターフェイスのことです。

解説資料としては、第 53 回東京エリア Debian 勉強会の資料 (`http://tokyodebian.alioth.debian.org/2009-06.html`) でまとめられています。今回は、この資料を参照しながら、ハンズオンとして実際に DDTSS を使った翻訳作業をやっていきます。

9.2 事前課題の確認

今回は、準備として事前課題の形をとってみましたが、いかがでしたでしょうか。

今後のセッションでも、適宜事前課題をからめていきたいと思っていますので、気づいたことがあれば教えてください。

9.2.1 DDTSS アカウントの作成

DDTSS はウェブサービスで、ユーザのアカウントを独自に保持しています。作業内容や実績はアカウントとひもづけて保管されるので、最初に作っておきましょう。[*20] ユーザ ID として利用される、メールアドレスが必要になります。

では作成したアカウントで、DDTSS にログインしてみてください。作成がまだの人はここで作ってしまいましょう。

9.2.2 気になるパッケージを探す

DDTSS の翻訳で扱う「パッケージの説明文」は、個別にはあまり量もないですし、翻訳作業をするにあたって、とっかかりの敷居は低い方だと思います。

とはいえ、見たことも聞いたこともないパッケージの説明というのもやっぱり辛いものがありますので、まずは関心があるとか、使っているとか、見たことある、といったものから手をつけてみるといいでしょう。

9.3 レビューをしてみる

DDTSS では、誰かが訳した一つの説明文は、オフィシャルの配布データにとりこまれるまでに、3 人からレビューを受けて「修正なしで OK」とされる必要があります。

レビュー 3 人というのは、実は結構高いハードルで、ついついレビュー待ちのものがたまってしまいがちです。

[*20] DDTSS create login `https://ddtp.debian.net/ddtss/index.cgi/createlogin`

そういうわけで、いきなり翻訳するというのはしんどいなぁ、という方も、ひとまずレビューにチャレンジしてみましょう。すでに翻訳された文章を確認するだけなので、敷居は低いと思います。

この文章を書いている時点では、レビュー待ちの数がずいぶん少ないので、選ぶのに困ってしまうかも知れませんが……。

9.3.1 レビュー待ち一覧の見方

DDTSS のログイン画面を、下の方にスクロールしてみてください。「Pendingreview」と書かれたセクションがあるはずです。下に並んでいるパッケージが、レビューを待っているもので、「Pending review」の横にある数字はその合計数になります。各パッケージ名の横を見ると、レビューの状況がわかります。

「needs initial review」は、翻訳された後まだ一回もレビューされていないということです。

「needs review had x」は、レビュー途上にあるもので、x がレビューされた回数を示しています。

9.3.2 レビュー画面

ではパッケージを選んでみましょう。レビュー画面に切り替わります。いくつかのメタ情報と合わせて、パッケージの説明文が、原文と対比される形で表示されているはずです。

まず、訳文を読んでみてください。意味は通りますか？ 誤字はありませんか？おかしいな、と思ったところは、原文と見比べてみてください。

レビューしてみて、特に訂正が必要ない状態であれば、「Accept as is」ボタンを押します。これで、レビューカウントが 1 つ増えることになります。

これはどう見てもおかしいな、というところがあれば、フォームの上で編集して、「Accept with changes」ボタンを押します。訳文は変更されて、レビューは振り出しに戻ることになります。

ちょっと違う気がするんだけど、いまいち自信がない……という場合は、コメント欄 (comment field) がありますので、そこに書いて、「Change comment only」ボタンを押します。この場合、レビューとしては何も変更されず、コメントだけ追記されることになります。

開いてみたけど、やっぱりレビューをやめる、という場合は、何もボタンを押さずに戻ってください。

9.3.3 実際にレビューしてみる

少しまとまった時間をとりますので、レビューにチャレンジしてみてください。わからないことがあれば、適宜呼んでください。

9.4 最初の翻訳文作成

いくつかレビューをしてみて DDTSS にも慣れてきたら、翻訳にも手を出してみてください。翻訳されていないパッケージ説明文は、「Pending Translation」の一覧にあります。気になるものを選んでみましょう。

画面の構成は、レビューとほとんど同じですが、翻訳文は、あったりなかったりします。翻訳文がある場合でも、原文が更新されていたり、似ている別のパッケージからひっぱられたりしているので、何らかの修正は必要になるはずです。

実際の翻訳画面にも赤字で表示されますが、やっぱり翻訳をやめる、という場合には「Abandon」というボタンを押すことになっているので注意してください。

ここは、時間が許せば実際にやってみようと思います。

10　DDTP 翻訳作業での TIPS

あけど

10.1　はじめに

Debian Package をインストールする際に参考する情報、**Description** 昔はすべて英語でしたが、最近はすこしづつ日本語になってきているのはご存知でしたか? Debian Description Translation Project は Debian Package の Description 文 (パッケージの内容を説明する文章) を日本語にするためのプロジェクトです。[*21]

日本語の翻訳の進捗は図 10.1 の状況です。

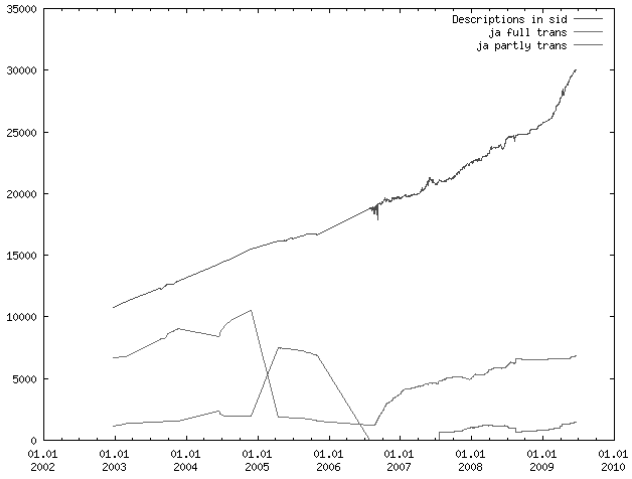


図 10.1　日本語の状況

前回、5 月の東京エリア Debian 勉強会でトピックに挙っていました DDTSS について、作業上の留意点などをまとめてみました。DDTSS については運用ルールなどで明確になっていない部分もありますので、その議論を深める材料になればと思います。このセクションはこれから翻訳を始めてみようという方へのガイドとしても考慮したつもりです。なお、このセクションにおいての幾つかの見解は筆者の個人的見解が多分に含まれていますので、その点ご留意頂ければと存じます。

`http://tokyodebian.alioth.debian.org/pdf/debianmeetingresume200905.pdf` p.31「8 DDTSS 活用」

10.2　ガイド等の文書の把握

5 月の資料にまとまっていますが、その後に作成されたページをご案内します。

[*21] DDTP サイト: `http://ddtp.debian.net/`

- DDTSS_ja/Faq - Debian Wiki http://wiki.debian.org/ja/DDTSS_ja/Faq

以下は 5 月の再掲です。

- Debian パッケージの説明文を日本語で読みたい! ～DDTP へのお誘い～http://www.debian.or.jp/community/translate/description-ja.html
- 武藤健志さんの blog の『Debian ドキュメント翻訳手続き』: http://kmuto.jp/d/index.cgi/debian/debian-doc-procedure.htm
- 小林儀匡さんの Debian 勉強会 2006 年 9 月資料「翻訳への誘い」: http://tokyodebian.alioth.debian.org/pdf/debianmeetingresume200609.pdf
- debian-doc メーリングリスト: 主要な議論が行われています。質問なども、こちらで。

10.3 翻訳 (辞書) サイトの利用

翻訳するのに十分な知識があればそもそも辞書は必要無いのかも知れません。印刷された辞書を使うのはインターネット接続がない環境では有効ですが、コンピュータを使う現代的な環境であればインターネット接続はほぼ不可欠でしょうから、オンラインの翻訳サイトを利用しない手はありません。以下にお勧めのサイトをご紹介します。

- 英単語辞書
  - 英辞郎 on the web スペースアルク http://www.alc.co.jp/ 収録単語の量、質、訳がそれぞれ秀逸です。スラング等の現代語から専門用語まで幅広く、また対象の英単語が含まれる例文が実例に基づいていて実用的な訳を得られます。(※このサイトを利用しているだけで、中学、高校で受けた英語が英文学を主眼に置いたものだと実感しました。)
- 文章翻訳
  - Yahoo!翻訳　http://honyaku.yahoo.co.jp/ 文単位での訳と原文の対応が付くので、長い文章や複雑な文章の翻訳に向いています。
  - 英語翻訳 - エキサイト 翻訳 http://www.excite.co.jp/world/ 自然な表現の日本語訳を得られます。カタカナ語の訳語をかなり上手く当てはめてくれたりします。
  - Cross Translation::色々な翻訳サイト・翻訳サービスの一括、横断翻訳 http://sukimania.ddo.jp/trans/trans_e.php 上記の Yahoo!翻訳、エキサイト 翻訳などの翻訳サービスにて一括で翻訳するサービスです。翻訳に迷った時の参考になるでしょう。但し、全てのサイトでの結果を得るには多少時間がかかります。

  他にも辞書・翻訳サイトは多数ありますので、各自で探してみて下さい。

10.4 Google で 「Debian」+「対象のパッケージ名」を検索

Debian パッケージの過去バージョンで既に翻訳されている事がありますのでそれを参考にすることもできます。(注: DDTSS に訳が取り込まれていない場合があるのです。)

10.5 分かりやすい文章にするための幾つかの手法

- 直訳の表現が日本語的に分かり難い場合

  意訳した方が分かりやすいならその方が望ましいです。
- ドイツ系の人が書いた英語で、関係代名詞が多用されていて直訳では日本語として分かり難い文章になることがよくあります。

  そういった場合には、文を分かりやすいサイズに区切りましょう。
- 専門用語と思われる訳語が直訳では意味が分かりない事があります。

専門用語はその分野での良く使われる訳語の表現を調べるようにしましょう。
(注:下記に注意点をまとめました)

10.6 DDTSS 限定 TIPS

- コメント欄の活用
  そのパッケージの翻訳に関しての注意事項などをレビューする人に向けて書いておくとレビューの際の参考になります。専門用語や意訳など、普通とは違う翻訳をした場合などにその理由などの根拠を書いておくと次の段階のレビュー作業をする人に理解してもらいやすくなります。
- Accept with changes がクリックできない場合
  翻訳文だけを更新した時に Accept with changes がグレーアウトしたままでクリックできないときがあります。タイトル部を変更することで対応できます。以下はその手順です。
  1. タイトルに空白等を追加する。
  2. Tab キーで次フォームにカーソルを移動すると、タイトルが更新されたとみなされ Accept with changes がクリック出来るようになります。
  3. タイトルに追加した空白等の文字を削除し、元に戻します。
  4. Accept with changes をクリックします。

10.7 専門用語の扱い

- 専門用語の見分け方
  決め手はありません。パッケージの説明に記述された関連分野から推測するか、その分野について (必要に応じて) 調べる事を念頭に置いて作業するのが良いでしょう。
- パッケージ説明で良く見かける専門分野とその専門用語及び良く使われている訳語 [*22]
  - 生物学、バイオインフォマティックス (biology, bioinfomatics)
    DNA 解析に関するパッケージが多数あります。
    annealing: アニーリング、(遺伝アルゴリズムの焼きなまし法 (Simulated Annealing)、機械加工の熱処理とは別のもの)
    align, alignment: アライン、アライメント、整列
    multiple alignment: マルチプルアライメント
    multiple sequence alignment: マルチプルシークエンスアライメント、(複合配列整列)
    polymerase chain reaction: ポリメラーゼ連鎖反応、略語 PCR、ポリメレースチェーンリアクション
    sequence: シークエンス、(遺伝子などの) 配列
  - アマチュア無線 (hamradio) band: 周波数帯、バンド
    CW: 電信
    modem: モデム
    Sporadic E layer: スポラディック E 層 (sporadic を突発的と訳さないのが一般的です。)
    N meter band: N メーターバンド (波長を周波数帯の別名として呼ぶのはアマチュア無線では一般的です。)
  - その他、複数の分野にまたがる用語
    Markov chain: マルコフ連鎖　統計学、数学、物理学、統計力学、情報科学

[*22] この段落付近は筆者の経験に大きく依存しています。

10.8 専門用語の訳についてのルール

どう扱うかのルールは未だ決まっていません。
　現状の対応
各々の分野のサイトでの表記を参考にしています。
良く使われる表記が複数ある場合はケースバイケースで妥当と思われるものを採用しています。
例) muliple sequence alignment の訳語

1. 多重配列整列
2. 多重配列アライメント
3. マルチプルシークエンスアライメント

この様な専門用語について今まで耳にした意見を挙げてみます。

1. 日本語より英語での表現が一般的であれば訳さない方が利用者にとって便利なので、訳さないかせいぜいカタカナにして持ってくるのが良い。例) CPU, メモリ、DHCP,etc
2. 日本語の訳が複数あり、利用状況がまちまちであるなら上記の案を勘案してパッケージ検索時の利便性を重視して、日本語の表記に () 括弧内に原文の語を示すのはどうか？
   例) muliple sequence alignment
   1. 多重配列整列 (muliple sequence alignment)
   2. 多重配列アライメント (muliple sequence alignment)
   3. マルチプルシークエンスアライメント (muliple sequence alignment)

   日本語の表記にばらつきがありますが、パッケージ検索で意図した内容は実現できそうです。ただし、この様な語句が多いと見やすさを損なうかもしれません。
3. その他、ご意見募集中

10.9 最後に

記述にあたって、様々な方々との (会話なども含む) やり取りを参考にしました。翻訳サイトの案内については吉田@板橋さんの小江戸らぐでの発表を参考にしました。DDTSS での作業についての助言や、このセクションの構成については、やまだたくまさんからのアドバイスを参考にしました。また、他の皆さんからのご意見が大変参考になりました。そういったものもできるだけ盛り込んだつもりです。DDTSS により翻訳作業のしきいは大きく下がったと思います。様々な人が少しでも多く DDTSS で翻訳作業に参加できるようになれば、作業の量だけでなく質も向上させることができると期待できます。

10.10 追加：第 53 回東京エリア Debian 勉強会で出された意見

- 日本語としての自然さなど、翻訳の質が一定しないのはどうかと思う。
  → 何か基準があれば良い？
  → でも、基準を作るのは難しいし、それより沢山の人が色々な観点からレビューする方が有効ではないか？
- 翻訳のコツとして心がけていること
  - 直訳より、日本語として分かり易い文章になるようにする。
  - 長文は短文に分割する。
  - 形容詞が連続する部分は (強調の意味が大きいので) 意図的に訳さない。
- DDTSS 利用上の注意点
  ログインしなくても翻訳できてしまうので必ずログインするようにしましょう。

- 専門用語について
  下手に日本語に訳されていると検索で見つからないので、敢えて訳さずに原文のままの方がいい
  → LANG=C aptitude search 〜 とすると英語での検索になるが、それではどうか？
  専門用語の訳語が複数あって適用される状況がまちまちな場合があるが、どうするのが良いか？
  → ガイドラインを設定するのはどうか？
  → ガイドラインを用意するのは難しいし、それより適用事例をまとめたものを用意した方が使いやすい
- 翻訳が未経験または自信の無い人はレビューから始めたらどうか
- OmegaT のような翻訳ツールを活用して翻訳辞書 (翻訳事例) を共有できると、翻訳の質が安定するのでは？
  → OmegaT についてはもっと情報を集める必要がある

11 DDTSS の利用方法の紹介

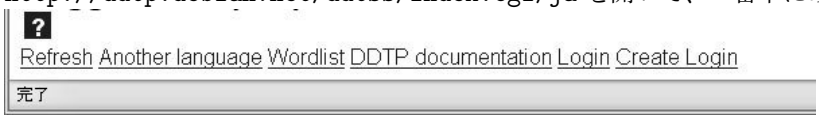
やまだたくま

11.1 はじめに

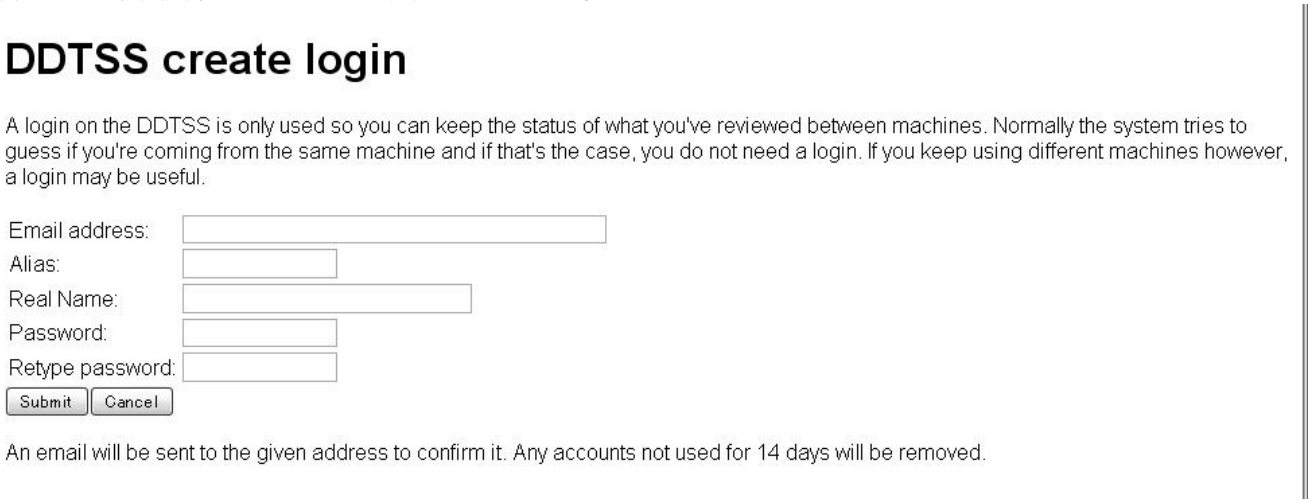
Debian パッケージの説明文を翻訳するプロジェクト DDTP にある Web インタフェース DDTSS の使い方を簡単にご紹介します。

11.2 アカウント作成とログイン

`http://ddtp.debian.net/ddtss/index.cgi/ja` を開いて、一番下にある Create Login をクリックします。



E メールアドレス (Email Address) とログイン名 (Alias)、本名 (Real Name)、パスワードを 2 回入力 (Password, Retype Password) して Submit ボタンを押して送信すると、アカウント作成は完了です。Alias は各種ログの表示で使われます、英文字のわかりやすい表記にしましょう。



アカウントを作成したらログインします。作業画面の右上にアカウントの情報が表示されます。

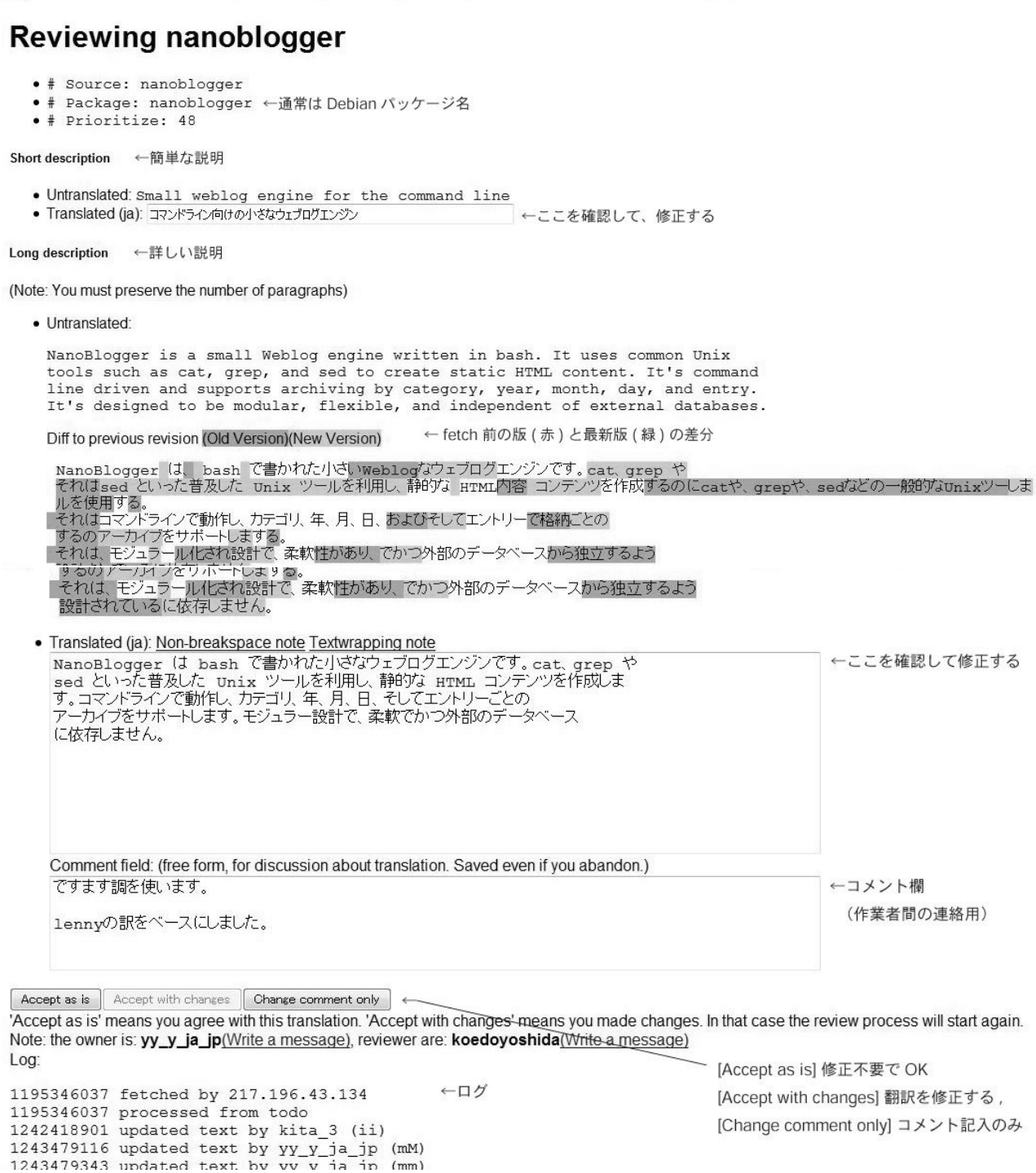
11.3 DDTSS で行ってほしい作業 - レビュー

現在 DDTSS でもっとも必要な作業は「レビュー」です。レビューの対象は、次の場所に表示されています。

```
Pending review (98)

   1. bristol (needs initial review)
   2. pngquant (needs initial review)
   3. adesklets (needs initial review)
   4. inventor-demo (needs initial review)
   5. emboss (needs review, had 1)
```

パッケージ名をクリックすると内容が表示されます。その内容を見て、簡単そうならレビューをします。難しいと思ったら、次のパッケージへ飛ばしてかまいません。



Raw form:

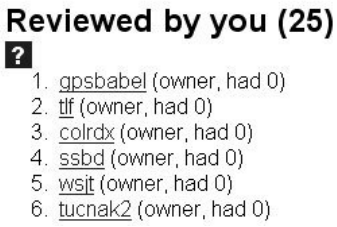
```
# Source: nanoblogger          ←DDTP ( メールインターフェイス ) 形式
# Package: nanoblogger
# This Description is active
# This Description is owned
# Prioritize: 48
Description: Small weblog engine for the command line
 NanoBlogger is a small Weblog engine written in bash. It uses common Unix
 tools such as cat, grep, and sed to create static HTML content. It's command
 line driven and supports archiving by category, year, month, day, and entry.
 It's designed to be modular, flexible, and independent of external databases.
Description-ja: <trans>
 <trans>

# other Descriptions of the nanoblogger package with a translation in ja:
#
# Description-id: 10475 http://ddtp.debian.net/ddt.cgi?desc_id=10475
# patch http://ddtp.debian.net/ddt.cgi?diff1=10475&diff2=39759&language=ja
# This Description was in sid from 2005-07-16 to 2007-04-17;
# This Description was in etch from 2005-07-16 to 2007-11-18;
#
```

11.4 レビューする

翻訳内容に問題ないと判断できた場合は、[Accept as is] ボタンをクリックしてください。

レビューをすると、表示される場所が変わります。さらにレビューが必要な場合は、次のようになります。

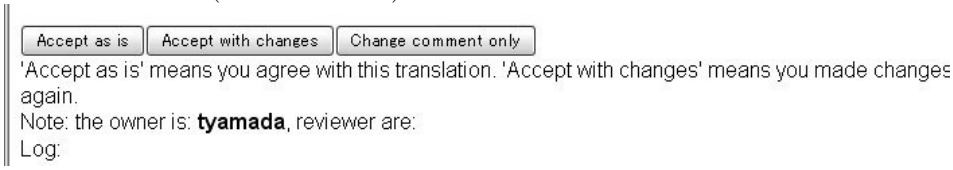


連続して 3 人がレビューをすると、翻訳作業が完了します. 完了後は次のようになります。

途中で誰かが翻訳を修正したら、再び 3 人のレビューが必要になります。

11.5 レビューを修正する

翻訳内容にすぐ修正が必要なら、テキストを変更してから [Accept with changes] ボタンをクリックしてください。なるべくコメント欄 (Comment field) に変更内容を記入してください。



11.6 翻訳を新規に作成する

翻訳を新規に作成する場合は　Pending Translation から選択してください。レビューの際と違うのは、選択肢が Abandon (翻訳を中止する) と Submit (翻訳を確定しレビューにまわす) の二つになっている点です。

11.7 debian-doc ML へのお誘い

翻訳の内容に疑問があったり、相談が必要だと思ったり、レビューや修正で (小さくない) 変更があったら、debian-doc ML に流してください。

debian-doc ML の宛先は、`debian-doc@debian.or.jp`

参加方法は、`http://www.debian.or.jp/community/ml/openml.html#docML`

質問や相談はフリーフォーマットです。

レビューを依頼するときは、次のメール例を参考にしてください。

(メールの例)

```
Subject: DDTSS レビュー icedax

xxx です。

xxxx さんが修正した icedax をレビューしました。


Short description

原文: Creates WAV files from audio CDs
訳文: オーディオ CD から WAV を生成する

Long description

原文:
icedax lets you digitally copy ("rip") audio tracks from a CD, avoiding
(略)

訳文:
icedax により、CD からディジタルでオーディオトラックのコピー ("リッピング")
(略)

用語を一部修正しました。
ディジタル->デジタル

修正後:
icedax により、CD からディジタルでオーディオトラックのコピー ("リッピング")
(略)

修正処理をしましたので、内容の確認をお願いします。
```

11.8 debian-doc ML だけでレビューする

「DDTSS をやる時間がないよ」という方は、debian-doc ML に流れているレビューや翻訳完了のメールを確認して、何か問題が見つけたらメールに返信してください。きっと誰かが DDTSS に反映するでしょう。

11.9 最後に

DDTSS は仕組み上レビューが大量に必要になります。多くの人が参加していることが前提になっています。なので、翻訳が正しい内容なのかを利用者の視点でレビューしてくれると助かります。

12 GUI がついて格好良くなった reportbug を使ってみよう

のがたじゅん

12.1 はじめに

Debian のバグ報告支援ツール reportbug が、バージョン 3.99.0 から GTK2 を利用したユーザーインターフェースが使えるようになっていましたが、あまり知られていないので reportbug の使い方も兼ねて書いてみました。

12.1.1 おわび

「使ってみよう」とタイトルに書きましたが、実際に GTK2 インターフェースで使うと querybts がうまく動かず登録されたバグを表示することができなかったり、戻るボタンで戻ると入力ができなかったりと不具合あります。ですので「使う」には難しいところもありますが、とりあえずこんな感じということを知ってもらえれば思います。

12.2 reportbug とは

Debian を使っていて発見したバグは、決められた書式にしたがって書いたメールを Debian のバグ追跡システム (Bug Tracking System. 通称 BTS) に送ることによってバグを報告することができます [*23]。ですがバグ報告に慣れていないと、決められた書式でメールを書くことはもちろんのこと、バグ修正に必要なパッケージ情報などを洩れなく集めたりすることは大変なことです。

reportbug は、そういったバグ報告の負担を減らすためのツールで、バグ検索から、バグ報告のためのパッケージ情報の収集、バグ登録に必要なメールの書式整形、報告メールの送信まで行ってくれるツールです。

12.3 reportbug のインストール

Debian インストール時に「標準システム」を選択すると、reportbug はインストールされますが、GTK2 や端末上でのメニューインターフェースに必要な Recommends や Suggests パッケージはインストールされないので、改めて reportbug をインストールします。

```
# apt-get update
# apt-get remove reportbug
# apt-get -o APT::Install-Suggests=true --install-recommends -y install reportbug
```

まどろっこしいことをしていますが、すでにインストールされたパッケージの Recommends や Suggests パッケージをスマートにインストールするにはどうしたらいんでしょうか?

[*23] バグレポートから参加する Debian パッケージ開発 / 木下 達也 (あんどきゅめんてっど でびあん 2008 年夏号)
http://tokyodebian.alioth.debian.org/pdf/debianmeetingresume2008-natsu.pdf

12.4 reportbug を起動

reportbug を起動します。

デスクトップのメニュー ([アプリケーション] → [システムメニュー] → [reportbug]) から起動すると、GTK2 インターフェースのウィンドウが開きます。端末から起動するとメッセージが表示されます。

```
$ reportbug
```

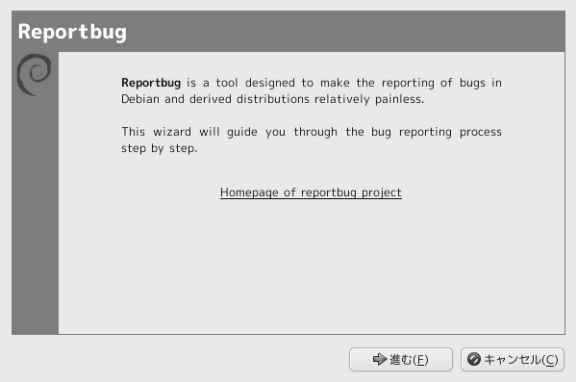


図 12.1 reportbug の GTK2 インターフェース

12.5 初期設定

reportbug を初めて起動すると、reportbug を使うための設定を尋ねられます。初期設定の途中、バグ報告メールを送信するためのメール環境について尋ねられるので、あらかじめメールの設定を確認しておいてください。設定例では、Gmail を使っていることを前提に話を進めます。

- 参考: その他のメール クライアントの設定 - Gmail ヘルプ:
  `http://mail.google.com/support/bin/answer.py?hl=jp&answer=13287`

```
Welcome to reportbug! Since it looks like this is the first time you have used reportbug, we are configuring its behavior.

These settings will be saved to the file "/home/hoge/.reportbugrc", which you will be free to edit further.
Please choose the default operating mode for reportbug.

1 novice    Offer simple prompts, bypassing technical questions.

2 standard  Offer more extensive prompts, including asking about things that a moderately sophisticated user would be
            expected to know about Debian.

3 advanced  Like standard, but assumes you know a bit more about Debian, including "incoming".

4 expert    Bypass most handholding measures and preliminary triage routines. This mode should not be used by people
            unfamiliar with Debian's policies and operating procedures.

Select mode: [novice]
```

ウェルカムメッセージと設定は /.reportbugrc に保存されていますと二つのお知らせの後、reportbug で使うモードについて尋ねています。とりあえずは細かいことよりバグ報告だけに絞りたいので、novice のままでよいでしょう。

```
Please choose the default interface for reportbug.

1 text   A text-oriented console user interface

2 gtk2   A graphical (GTK+) user interface.

3 urwid  A menu-based console user interface

Select interface: 2
```

どのユーザーインターフェースを使うか選択します。今回はテーマにそって gtk2 を選びました。実際に使うとなると urwid が使いやすいと思います。

```
Will reportbug often have direct Internet access? (You should answer yes to this question unless you know what you are doing
and plan to check whether duplicate reports have been filed via some other channel.) [Y|n|q|?]?
```

インターネットに接続しているかですが、おそらくほとんどの人は接続していると思うので、y と答えます。

```
What real name should be used for sending bug reports?
[Hogewo HOGETA]>
```

バグ報告に使う名前を設定します。

```
Which of your email addresses should be used when sending bug reports? (Note that this address will be visible in the bug
tracking system, so you may want to use a webmail address or another address with good spam filtering capabilities.)
[example@gmail.com]>
```

バグ報告に使うメールアドレスを設定します。ここで気をつけないといけないことは、ここで設定したメールアドレスはバグ報告をすると公開されます。あまり公開したくないメールアドレスは設定しないほうがよいでしょう。

```
Do you have a "mail transport agent" (MTA) like Exim, Postfix or SSMTP configured on this computer to send mail to the
Internet? [Y|n|q|?]? n
```

バグ報告をするマシンでメールサーバーが動いているか尋ねています。Gmail の SMTP サーバーを使って送信するので、ここでは n と答えます。

```
Please enter the name of your SMTP host.  Usually it's called something
 like "mail.example.org" or "smtp.example.org".
If you need to use a different port than default, use the <host>:<port> alternative format.

Just press ENTER if you don't have one or don't know.
> smtp.gmail.com:587
```

送信する SMTP サーバーを登録します。Gmail の SMTP サーバーのアドレスは smtp.gmail.com、ポート番号は 587 なので、smtp.gmail.com:587 と登録します。

```
If you need to use a user name to send email via "smtp.gmail.com:587" on your computer, please enter that user name. Just
press ENTER if you don't need a user name.
> nogajun@gmail.com
```

SMTP サーバーを利用するためのユーザー名を登録します。Gmail の SMTP サーバーを使うにはドメイン名も含めたユーザー名が必要なので、¡ユーザー名¿@gmail.com と登録します。

```
Does your SMTP host require TLS authentication? [y|N|q|?]? y
```

SMTP は TLS を使うかの問いについては、Gmail では必要なので y と答えます。

```
Please enter the name of your proxy server. It should only use this parameter if you are behind a firewall. The PROXY
argument should be formatted as a valid HTTP URL, including (if necessary) a port number; for example,
http://192.168.1.1:3128/. Just press ENTER if you don't have one or don't know.
>
```

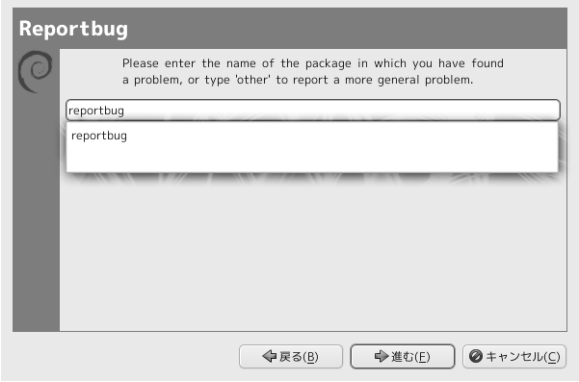
Proxy について尋ねています。使っていれば設定します。使っていなければ空のままで Enter キーを押します。

```
Default preferences file written. To reconfigure, re-run reportbug with the "--configure" option.
```

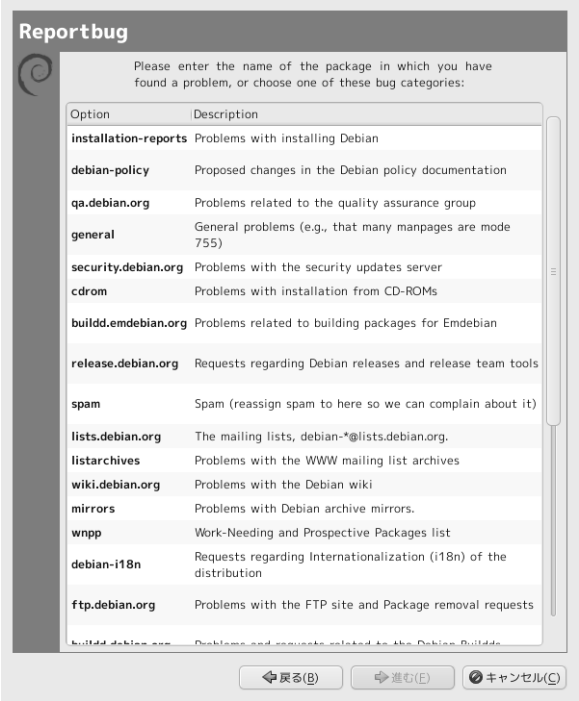
初期設定が終わりました。もう一度、設定をやり直したいときは、–configure オプションをつけて起動するとやり直すことができます。

12.6 reportbug を使ってみる

起動するとこんな感じです。ウィザード形式なのでわかりやすいと思います。



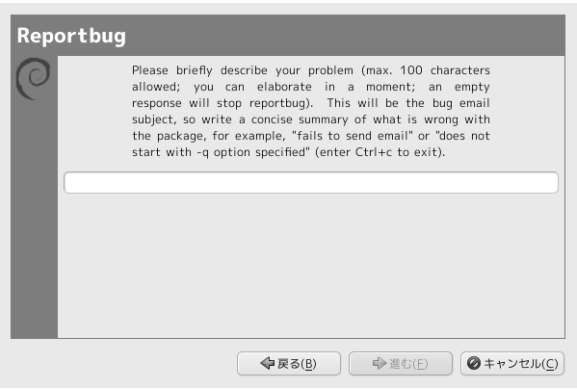
バグを見つけたパッケージ名を入力します。パッケージ名の頭を何文字か入力すると、インストールされているパッケージの候補を表示してくれます。



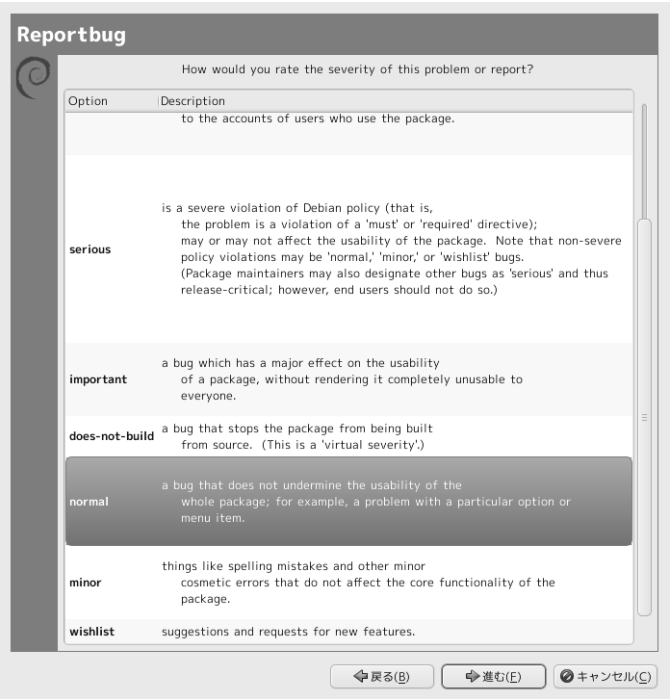
パッケージ名の入力で other と入力するとこの画面になります。これは擬似パッケージの一覧です。パッケージ以外に見つけた Debian に関連する問題を擬似パッケージのバグとして登録します。

- Debian – Debian バグ追跡システム - 擬似パッケージ:
  `http://www.debian.org/Bugs/pseudo-packages.ja.html`

本来ならばここに登録されたバグが表示されて、今から登録しようとしたバグがすでに登録済みかどうか確認できるのですが、GTK2 インターフェースだと querybts がうまく動かなくて (#432178)、ダブルクリックをしても何も表示されません。残念。



バグがまだ登録されていなかったとして次に進むと、バグを登録する画面に入ります。ここではバグの概要を簡潔に書きます。



バグの種類を指定します。種類は Critical や Serious のような深刻なものから、提案の Whishlist まであります。通常は normal でいいと思います。

ここでは、バグの内容を書きます。スクリーンショットのために適当なことを書いて撮っていたので Subject は消していますが、通常は Subject の箇所に上で書いたバグの概要が入っています。

バグの内容を書き終わって進むと、この画面になります。

この時点ではまだメールは送信されてません。問題がないと思うなら、Submit the bug report via email. ボタンを押して送信します。いや、もう少しだけ確認したい、とりあえず下書きで保存して続きは後で書く、ファイルを添付したいなどあれば、それぞれのボタンを押して終了します。

12.7 まとめ

reportbug を使うと簡単にバグ報告ができるので、みんなバグを見つけたらバグ報告しよう。

あと、のがじゅんはさっさと仕事をしよう。実例をあげて説明しようと目星をつけてたパッケージを見たら、バグが修正されてて困った困った。

12.8 参考資料

- Debian JP Project - バグ報告:
  `http://www.debian.or.jp/community/bugreport.html`
- Debian – Debian BTS - バグを報告する:
  `http://www.debian.org/Bugs/Reporting.ja.html`
- MC-MPI Debian 公式パッケージへの道 / 藤澤 徹
  第 52 回東京エリア Debian 勉強会 2009 年 5 月勉強会 配布資料
  `http://tokyodebian.alioth.debian.org/pdf/debianmeetingresume200905.pdf`

13 Debian パッケージの作り方。最初から最後まで

佐々木洋平

13.1 はじめに

... なんて**大それた**タイトルなんでしょう.. 私事で忙しくて訂正できなかった訳ですが、 我ながら恥ずしいです。
さて、 佐々木のパッケージ作成遍歴は以下の通りです:

1. 売り物ソフトウェアを dpkg で管理するために弄り始める
2. 自分達の作っているソフトウェアの deb パッケージを作成し始める。
3. どうせなら本家に... ← イマココ

今日のお話は、 これらを踏まえての「パッケージ作成最初から最後まで」です。ここでは「最初」を「ソースの取得」、「最後」を「lintian & piuparts clean」とします。個々の How to、 特に一番ハマりやすい `debian/rules` については時間の紙面の都合上参考文献へのポインタを示すに留めます。 是非質問して下さい。

13.2 Package

コンパイル後のソフトウェアなどをすぐ利用できる形にまとめたものをバイナリパッケージと呼びます。Debian では拡張子が `.deb` のファイルがこれにあたります。我々は普段 `apt-get` や `aptitude` を利用して、 バイナリパッケージを導入/更新/削除したりしています。バイナリパッケージは制御情報とデータを `tar.gz` に圧縮し、バージョン情報とともに `ar(1)` でまとめたものです。これらは非常に一般的なコマンドですから、バイナリパッケージを展開するだけならば多くのシステムで可能です。

バイナリパッケージに対して、これを作成するための素材をまとめたものをソースパッケージと言います。これは二つないし三つのファイルからなります:

- オリジナルのソース一式 (`.orig.tar.gz`)
- パッケージの情報 (`.dsc`))
- バイナリパッケージを作成するための変更 (`.diff.gz`)
  - オリジナルのソースが無い場合 (Debian 固有のパッケージ等) の場合は存在しない。

ソースパッケージは導入したり削除したりする性質のパッケージではありません。目的はバイナリパッケージの作成にあります。Debian が提供しているバイナリパッケージには、対応するソースパッケージが必ず存在しており、必要に応じてソースパッケージを取得してバイナリパッケージを構築することができます。

13.2.1 deb package inside

dpkg-deb コマンドを利用して、 実際にパッケージを展開してみましょう。例えば rabbit[*24] というパッケージの deb ファイルを展開してみると...

```
% dpkg-deb -x rabbit_0.6.1-1_all.deb rabbit
            (rabbit というパッケージを rabbit というディレクトリに展開)
% dpkg-deb -e rabbit_0.6.1-1_all.deb rabbit/DEBIAN
            (rabbit パッケージの制御情報を rabbit/DEBIAN に展開)
% cd rabbit ls
DEBIAN/  usr/
% tree
.
|-- DEBIAN
|   |-- control
|   |-- md5sums
|   `-- preinst
`-- usr
    |-- bin
    |   |-- rabbit
-------- snip ---------------
    |   `-- rabbit-theme-manager
    |-- lib
    |   `-- ruby
    |       `-- 1.8
    |           |-- rabbit
-------- snip ---------------
    |
    `-- share
        |-- doc
        |   `-- rabbit
        |       |-- NEWS.en.gz -> changelog.gz
-------- snip ---------------
        |       |-- README.Debian
        |       |-- README.en.gz
        |       |-- README.ja.gz
        |       |-- changelog.Debian.gz
        |       |-- changelog.gz
-------- snip ---------------
        ...
179 directories, 575 files
```

パッケージの制御情報は DEBIAN 以下に展開しました。rabbit の場合、

```
% ls -R DEBIAN
DEBIAN:
control  md5sums  preinst*
```

の三つからなります。 これらは

control　メンテナの名前、対応するソースパッケージ名、 依存関係などが記述されたファイル

md5sums　提供される各ファイルの md5 checksum

preinst　インストール作業の前に実行される hook シェルスクリプト。パッケージによっては、preinst 以外に postinst、 prerm、postrm などが存在します。

パッケージのデータは usr 以下に展開しました。deb パッケージを導入した際には、 これらは /usr 以下に展開されます。

というわけで上記構成になったディレクトリツリーを用意して tar.gz で圧縮したりするとバイナリパッケージができあがります。

13.2.2 余談: dpkg-deb でパッケージを再構築

売り物の (ソースが取得できない) ソフトウェアを Debian のパッケージシステムで管理したい時に佐々木がよくやる手段は

- alien で rpm を deb に変換
- 上記 dpkg-deb -e|-x でファイルを展開
- 適切に配置、 修正

[*24] RD や Wiki フォーマットで記述したテキストをベースにしたプレゼンテーションツール。 この間 unstable に入りました!!

- dpkg-deb -b で再アーカイブ

として、 似非パッケージを作成することです。当然配布はできませんが[*25]例として、 大昔の Intel Compiler Ver.8 のパッケージ作成は以下の様にやっていました。

```
% tar xvzf l_cc_pc_8.1.028.tar.gz
% cd l_cc_pc_8.1.028
% rm -rf *64*
% sudo alien *.rpm
% rm *.rpm
% sudo chown $USER *.deb
% mkdir tmp
% dpkg-deb -e intel-icc8_8.1-29_i386.deb tmp/DEBIAN
% dpkg-deb -x intel-icc8_8.1-29_i386.deb tmp/
% echo DESTINATION=/opt/`ls tmp/opt` >> tmp/DEBIAN/postinst
% cat <<EOF >> tmp/DEBIAN/postinst
 for FILE in $(find $DESTINATION/bin/ -regex \
    '.*[ei](cc|fort|fc|cpc)$\|.*cfg$\|.*pcl$\|.*vars[^/]*.c?sh$' \
    2> /dev/null) do
     sed s@\@$DESTINATION@g $FILE > ${FILE}.abs
     mv ${FILE}.abs $FILE
     chmod 755 $FILE
 done
 for FILE in $(find $DESTINATION/bin/ -regex '.*[ei]cc' 2> /dev/null) do
     sed s@\@$DESTINATION@g $FILE > ${FILE}.abs
     mv ${FILE}.abs $FILE
     chmod 755 $FILE
 done
 for FILE in $(find $DESTINATION/bin/ -regex '.*[ei]cpc' 2> /dev/null) do
     sed s@\@$DESTINATION@g $FILE > ${FILE}.abs
     mv ${FILE}.abs $FILE
     chmod 755 $FILE
 done
 for FILE in $(find $DESTINATION/bin/ -regex '.*[ei]fort' 2> /dev/null) do
     sed s@\@$DESTINATION@g $FILE > ${FILE}.abs
     mv ${FILE}.abs $FILE
     chmod 755 $FILE
 done
 for FILE in $(find $DESTINATION/bin/ -regex '.*[ei]fc' 2> /dev/null) do
     sed s@\@$DESTINATION@g $FILE > ${FILE}.abs
     mv ${FILE}.abs $FILE
     chmod 755 $FILE
 done
EOF
% dpkg-deb -b tmp intel-icc8_8.1-29_i386.deb
% dpkg -i intel-icc8_8.1-29_i386.deb
% dpkg -i intel-iidb8_8.1-46_i386.deb
% dpkg -i --force-overwrite intel-isubh8_8.1-29_i386.deb
```

最近は Intel Compiler に愛がない[*26]のでやっていませんが。

13.3 パッケージ作成

さて、 単にバイナリパッケージを作成するだけならば前小節 (13.2.2) で示した通り

- DEBIAN 以下に control、 md5sums、 必要ならば hook スクリプト
- パッケージとして / 以下に展開したいディレクトリ構成に揃えたファイル群

を作成して dpkg-deb でまとめれば良いだけです。ですが、 あんまり一般的ではありませんね。以下では、 GNU hello を例に[*27]、実際に配布まで含めたパッケージ作成方法について述べてみます。

13.3.1 前準備

■環境変数の設定 パッケージメンテナの名前とメールアドレスを環境変数に設定します:

```
DEBFULLNAME="Youhei SASAKI"; export DEBFULLNAME
DEBEMAIL=uwabami@gfd-dennou.org ; export DEBEMAIL
```

配布も考えているなら GPG 鍵の記述に合わせておくと良いと思います。

■最低限必要なパッケージの導入 build-essential メタパッケージを導入しておきます。 このパッケージは deb

[*25] 売り物って rpm ばっかりです。Ubuntu のおかげで大分減りましたが。
[*26] マイナーバージョン上がる度にディレクトリ構成がコロコロ変わるのでつきあいきれなくなりました
[*27] 「またー?」とか言わないこと

パッケージを構築するのに最低限必要となるパッケージを導入するメタパッケージです。具体的には

1. libc6-dev | libc-dev
2. g++
3. make
4. dpkg-dev

とこれに依存する幾つかのファイルが導入されます[*28]。

```
% apt-cache show build-essential
Package: build-essential
Priority: optional
Section: devel
Installed-Size: 48
Maintainer: Matthias Klose <doko@debian.org>
Architecture: amd64
Version: 11.4
Depends: libc6-dev | libc-dev, g++ (>= 4:4.3.1), make, dpkg-dev (>= 1.13.5)
Filename: pool/main/b/build-essential/build-essential_11.4_amd64.deb
Size: 7126
MD5sum: 86a942017ad93721c91212398a828a0c
SHA1: 5ac2ba90444e1eaed96b2163389a8812eb107b01
SHA256: 3dbd2e6b4e998412a6ad4d32b242523559b536168bcce7f227c7ce30256808a5
Description: Informational list of build-essential packages
 If you do not plan to build Debian packages, you don't need this
 package.  Starting with dpkg (>= 1.14.18) this package is required
 for building Debian packages.
 .
 This package contains an informational list of packages which are
 considered essential for building Debian packages.  This package also
 depends on the packages on that list, to make it easy to have the
 build-essential packages installed.
 .
----- snip -----
% sudo aptitude install build-essential
```

■ソースの取得、 確認 動かないソフトウェアをパッケージ化するのは大変ですよね? 事前にソースを取得して動作確認しておくと良いでしょう。 また「動作させるために patch を書いた!」という猛者は、 そのパッチを保管しておくと幸せになれるかもしれません。

GNU hello は次の URL から取得できます。http://www.gnu.org/software/hello/

GNU hello は configure ; make ; make install で導入するソフトウェアです。 実際に configure を動かしてみます。

```
% cd hello-2.4
% ./configure
...
```

エラーがでなければこれで終了です。もし./configure が必要なファイルを探せずエラー終了する場合には、apt-file コマンドで必要なファイルを提供している Debian パッケージを探してみましょう。

```
% sudo aptitude install apt-file
% sudo apt-file update
% apt-file search [file 名]
```

足りないファイルを導入したら、 もう一度 ./configure を走らせます。これをくりかえして、 必要なファイルを導入していきます。

無事 ./configure が通るようになったら **make** を実行してみます。

```
% make
make all-recursive
make[1]: Entering directory ‘/home/uwabami/Desktop/hello-2.4’
Making all in contrib
make[2]: Entering directory ‘/home/uwabami/Desktop/hello-2.4/contrib’
make[2]: Nothing to be done for ‘all’.
make[2]: Leaving directory ‘/home/uwabami/Desktop/hello-2.4/contrib’
...
make[2]: Entering directory ‘/home/uwabami/Desktop/hello-2.4’
make[2]: Leaving directory ‘/home/uwabami/Desktop/hello-2.4’
make[1]: Leaving directory ‘/home/uwabami/Desktop/hello-2.4’
```

[*28] dpkg-dev が make に依存しているので、なんか変な気分ですが、 正しいんですよね? きっと…

コンパイルも正常に終了したので、試しに実行してみます。

```
% ./src/hello
% ./src/hello -t
% ./src/hello -g "Good Night。.."
```

ここまでがパッケージ作成前の動作確認作業です。

実際に配布するためのパッケージを作成する場合には、動作確認以外にも Copyright や License を確認しておくべきです。

13.3.2 パッケージ作成

■雛形の作成 dh_make コマンドでパッケージの雛形を作成します。dh_make は、dh-make パッケージで提供されていますので、これを導入します

```
% sudo aptitude install dh-make
% dh_make --help
dh_make - prepare Debian packaging for an original source archive, version 0.50

Copyright (C) 1998-2009 Craig Small <csmall@debian.org>
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
  Usage: dh_make [options]
  -c, --copyright <type>    use <type> of license in copyright file
                            (apache|artistic|bsd|gpl|gpl2|gpl3|lgpl|lgpl2|lgpl3)
      --dpatch              using dpatch to maintain patches
      --quilt               using quilt to maintain patches
  -e, --email <address>     use <address> as the maintainer e-mail address
  -n, --native              the program is Debian native, don't generate .orig
  -f, --file <file>         specify file to use as the original source archive
  -r, --createorig          make a copy for the original source archive
  -s, --single              set package class to single
  -i, --indep                       set package class to arch-independent
  -m, --multi               set package class to multiple binary
  -l, --library             set package class to library
  -k, --kmod                set package class to kernel module
      --kpatch              set package class to kernel patch
  -b, --cdbs                set package class to cdbs
  -a, --addmissing          reprocess package and add missing files
  -t, --templates <dir>      apply customizing templates in <dir>
  -d  --defaultless         skip the default debian and package class templates
  -o, --overlay <dir>       reprocess package using template in <dir>
  -p, --packagename <name>  force package name to be <name>
  -h, --help                display this help screen and exit
  -v, --version             show the version and exit

By Craig Small <csmall@debian.org>
Based on deb-make by Christoph Lameter <clameter@debian.org>.
Custom template support by Bruce Sass <bmsass@shaw.ca>.
```

ここでは

–createorig, -r　オリジナルのソースファイル (.orig.tar.gz) を作成する

–copyright gpl, -c gpl　ソースのライセンスが gpl3 なので。ライセンスが代表的なモノの場合、指定しておくと雛形の時点で結構できあがっていて、 非常に楽です。

–single, -s　シングルバイナリのパッケージを作成します。ライブラリの場合には -l としてすると良いでしょう。

–cdbs, -b　CDBS(後述) を使用するので指定します。

–quilt, –dpatch　パッチを当てることが決まっているのであれば--dpatch もしくは--quilt を指定しておくと良いでしょう。今回は指定しません。

以下のコマンドを実行します。

```
% dh_make --createorig --copyright gpl --single --cdbs
(もしくは)
% dh_make -r -c gpl -s -b
```

実行すると以下のようなメッセージが表示されるので、確認して Enter を押します。

```
Maintainer name : Youhei SASAKI
Email-Address   : uwabami@gfd-dennou.org
Date            : Sun, 25 Oct 2009 00:08:43 +0900
Package Name    : hello
Version         : 2.4
License         : gpl3
Using dpatch    : no
Using quilt     : no
Type of Package : cdbs
Hit <enter> to confirm:
```

うまく動作すると、debian ディレクトリができ、このディレクトリ以下に雛形 (.ex, .EX) ができます。debian ディレクトリは以下のような状態になっています。

```
.
|-- README.Debian        (パッケージの README)
|-- changelog            (パッケージのチェンジログ)
|-- compat               (パッケージのバージョン)
|-- control              (パッケージ情報)
|-- copyright            (パッケージのコピーライト情報)
|-- cron.d.ex            (パッケージで cron を使う場合の設定ファイル)
|-- dirs                 (パッケージでデータを配置するディレクトリ名の設定)
|-- docs                 (パッケージに含めるドキュメントファイルを指定する)
|-- emacsen-install.ex   (emacs 用設定ファイル)
|-- emacsen-remove.ex    (emacs 用設定ファイル)
|-- emacsen-startup.ex   (emacs 用設定ファイル)
|-- hello.default.ex     (パッケージで debfonf を使う場合の設定ファイル)
|-- hello.doc-base.EX    (パッケージで doc-base を使う場合の設定ファイル)
|-- init.d.ex            (パッケージで init.d を使う場合の設定ファイル)
|-- init.d.lsb.ex        (パッケージで init.d を使う場合の設定ファイル)
|-- manpage.1.ex         (manpage の雛形)
|-- manpage.sgml.ex      (manpage の雛形)
|-- manpage.xml.ex       (manpage の雛形)
|-- menu.ex              (メニューの雛形)
|-- postinst.ex          (postinst メンテナファイルの雛形)
|-- postrm.ex            (postrm メンテナファイルの雛形)
|-- preinst.ex           (preinst メンテナファイルの雛形)
|-- prerm.ex             (prerm メンテナファイルの雛形)
|-- rules                (パッケージビルドスクリプト)
`-- watch.ex             (アップストリームチェック用ファイル)
```

ちなみにパッケージを作成する場合には**このディレクトリの中以外は触りません**。オリジナルのソースに変更を加える場合には quilt や dpatch 等のパッチシステムを利用すると良いでしょう。

今回はおもむろに .ex, .EX を削除します。

```
% rm -f debian/*.ex debian/*.EX
```

■CDBS　パッケージの実際の構築は debian/rules で行なわれます。debian/rules はいわゆる Makefile ですので、 make の文法で必要となる設定を行なっていきます。

GNU hello は./configure ; make ; make install で install しますのでこの場合は cdbs を使用した方が幸せになれます[*29]。dh_make に -b を指定した場合、 debian/rules は次の様になっています。

```
#!/usr/bin/make -f

include /usr/share/cdbs/1/rules/debhelper.mk
include /usr/share/cdbs/1/class/autotools.mk


# Add here any variable or target overrides you need.
```

include されているのは

- バイナリパッケージの作成支援のためのコマンドである debhelper を必要なタイミングで呼びだす。
- パッケージの作成に autotools を使う

という命令セットです。

CDBS の詳細については、例えば [CDBS 1st step]、[Online CDBS Gallery]、[CDBS Documentation Rev. 0.4.0] を参照下さい。

[*29] debhelper 7 の魔法の様な rules も凄いですが、 まだ使いこなせない...

■rules の調整　ここで一旦バイナリパッケージを作成してみましょう。

```
% sudo aptitude install fakeroot
% fakeroot debian/rules binary
...
```

rules の binary ターゲットを実行することで、ソースの一つ上のディレクトリにバイナリパッケージが作成されます。この時点ではバイナリパッケージには見向きもせず debian ディレクトリの下に生成される debian/hello 以下を確認します。

```
|-- DEBIAN
|   |-- control
|   `-- md5sums
`-- usr
    |-- local
    |   |-- bin
    |   |   `-- hello
    |   `-- share
    |       |-- info
    |       |   `-- hello.info
    |       |-- locale
    |       |   |-- bg
    |       |   |   `-- LC_MESSAGES
    |       |   |       `-- hello.mo
--------- snip ------------------
...
103 directories, 61 files
```

hello が /usr/local/bin に install されています。これは変ですよね? build 時の log を注意深く見ていると、configure 実行時に --prefix が指定されておらず、 /usr/local 以下に設定されています。

よって cdbs に対して configure 実行時に --prefix=/usr を指定するようにします。

```
#!/usr/bin/make -f

include /usr/share/cdbs/1/rules/debhelper.mk
include /usr/share/cdbs/1/class/autotools.mk

DEB_CONFIGURE_EXTRA_FLAGS:= --prefix=/usr
```

もういちど fakeroot debian/rules binary を実行します。これを繰り返して、 ファイルが望んだ配置になるまで debian/rules を修正していきます。

■制御情報の編集　具体的には

1. debian/control
2. debian/changelog
3. debian/copyright

の三つです。特記事項が無いならば debian/README.Debian は削除しても良いでしょう。

control の例:

```
Source: hello
Section: devel
Priority: optional
Maintainer: Youhei SASAKI <uwabami@gfd-dennou.org>
Build-Depends: cdbs, debhelper (>= 7), autotools-dev
Standards-Version: 3.8.3
Homepage: http://www.gnu.org/software/hello

Package: hello
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: The classic greeting, and a good example
 The GNU hello program produces a familiar, friendly greeting.  It
 allows non-programmers to use a classic computer science tool which
 would otherwise be unavailable to them.
 .
 Seriously, though: this is an example of how to do a Debian package.
 It is the Debian version of the GNU Project's 'hello world' program
 (which is itself an example for the GNU Project).
```

changelog の例:

```
% cat debian/changelog
hello (2.4-1) unstable; urgency=low

  * Initial release

 -- Youhei SASAKI <uwabami@gfd-dennou.org>  Sun, 25 Oct 2009 00:08:43 +0900
```

雛形には ITP のバグ番号を記述するところがあります。ITP している場合には埋めておくと良いと思います。

copyright の例:

```
This work was packaged for Debian by:

    Youhei SASAKI <uwabami@gfd-dennou.org> on Sun, 25 Oct 2009 00:08:43 +0900

It was downloaded from:

    http://www.gnu.org/software/hello/

Upstream Author:

Authors of GNU Hello.

  Copyright (C) 1999, 2005, 2006 Free Software Foundation, Inc.

  Copying and distribution of this file, with or without modification,
  are permitted in any medium without royalty provided the copyright
  notice and this notice are preserved.

The following contributions warranted legal paper exchanges with the
Free Software Foundation.  See also the ChangeLog and THANKS files.

Mike Haertel
David MacKenzie
Jan Brittenson
Roland McGrath
Charles Hannum
Bruce Korb              hello.c, configure.ac.
Karl Eichwalder         all files.
Karl Berry              all files.
The King                releases.

License:

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This package is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

On Debian systems, the complete text of the GNU General
Public License version 3 can be found in ‘/usr/share/common-licenses/GPL-3’.

The Debian packaging is:

    Copyright (C) 2009 Youhei SASAKI <uwabami@gfd-dennou.org>

and is licensed under the GPL version 3, see above.
```

ソースに AUTHORS とか COPYING とかある場合には、 編集が非常に楽ですね。

■debuild, lintian　バイナリパッケージとソースパッケージの作成、パッケージのポリシー違反の確認を行ないます。devscripts と linitian を導入します。

```
% sudo aptitude install debuild lintian
```

その後、 debuild コマンドを実行します:

```
% debuild -rfakeroot -uc -us
...
W: hello source: configure-generated-file-in-source config.status
W: hello source: configure-generated-file-in-source config.log
W: hello: new-package-should-close-itp-bug
Finished running lintian.
```

-uc、 -us は GPG サインをしない設定です。GPG でサインする場合にはこのオプションを省略して下さい。

最後に出てきたのが lintian によるチェック結果です。これを解決しましょう。最後の ITP は今回しょうがない

ですが、上の二つは config.status、 config.log を clean ターゲットが呼ばれた時に消去するようにすれば良いのです

```
% cat debian/rules
#!/usr/bin/make -f

include /usr/share/cdbs/1/rules/debhelper.mk
include /usr/share/cdbs/1/class/autotools.mk

DEB_CONFIGURE_EXTRA_FLAGS:= --prefix=/usr

clean::
        rm -f config.status config.log
```

この後でもう一度 debuild してみます。

```
% debuild -rfakeroot -uc -us
...
dpkg-deb: ‘../hello_2.4-1_amd64.deb’ にパッケージ ‘hello’ を構築しています。
 dpkg-genchanges  >../hello_2.4-1_amd64.changes
dpkg-genchanges: including full source code in upload
dpkg-buildpackage: full upload (original source is included)
Now running lintian...
W: hello: new-package-should-close-itp-bug
Finished running lintian.
```

ITP したならば、 debian/changelog に ITP の番号を書いておくと最後の warning も消せますね。

これでオシマイ。 メデタシメデタシ…ではありません。

13.4 パッケージのビルド・インストールテスト

まず、 ビルドテストを行ないます。ビルドテストは大雑把に言えば、ソースパッケージを元に

- debian/control の内容を元に構築に必要な他のパッケージを導入し
- debian/rules を実行してバイナリパッケージを作成し
- lintian に怒られないパッケージができあがる

ことをテストします。

ビルドテストには pbuilder を使用します。pbuilder は必要最小限の Debian パッケージが導入された環境の tar.gz を用いて、パッケージのビルド時にその tar.gz を展開し chroot してパッケージの作成を行ないます。

まずは pbuilder 環境を導入します

```
% sudo aptitude install pbuilder
% sudo pbuilder  --create --distribution sid
```

ちょっと時間がかかりますが気長に待ちます。これによって /var/cache/pbuilder/base.tgz が生成されます。これを使用してビルドテストを行ないます。

```
% sudo pbuilder --build --distribution sid --basetgz /var/cache/pbuilder/base.tgz hello_2.4-1.dsc
```

無事に build テストが通りましたか? きちんとパッケージができているなら/var/cache/pbuilder/result 以下にパッケージが置かれています。

さてパッケージのビルドテストが通ったら、次はインストール/アンインストールテストです。これには piuparts パッケージを使用します。

```
% sudo aptitude install piuparts
```

piuparts も pbuilder と同様に最低限の環境からインストール/アンインストールテストを実行します。 pbuilder で作成した base.tgz を使用してみましょう。

```
% sudo piuparts -d sid -b /var/cache/pbuilder/base.tgz hello_2.4-1_amd64.deb
...
0m50.1s DEBUG: No broken symlinks as far as we can find.
0m51.3s INFO: PASS: Installation, upgrade and purging tests.
0m51.3s DEBUG: Starting command: ['chroot', '/tmp/tmpZ2-nup', 'umount', '/proc']
0m51.3s DEBUG: Command ok: ['chroot', '/tmp/tmpZ2-nup', 'umount', '/proc']
0m51.7s DEBUG: Removed directory tree at /tmp/tmpZ2-nup
0m51.7s INFO: PASS: All tests.
0m51.7s INFO: piuparts run ends.
```

ここまできたらパッケージは一通り作成完了です。

13.5 まとめ

というわけで GNU hello を題材にパッケージ作成を最初から最後まで眺めてみました。実際には、 単一のソースから複数のバイナリパッケージを作成したり、ライブラリパッケージ (共有ライブラリ、 静的ライブラリ + ヘッダ、デバッグシンボル) を作成したり、 カーネルパッケージを作成したり、 と覚える事は一杯あります。ですが、 必要になったらその都度覚える、 で良いのではないでしょうか? 大丈夫です。 lintian **がちゃんと怒ってくれます**。

あと、 個人的には**魔法の様な debhelper** の使い方を取得したいです。例えば、 先日 unstable に入った libdap パッケージの debian/rules はこれだけなんですよ:

```
#!/usr/bin/make -f

DEB_CONFIGURE_EXTRA_FLAGS := --with-gnu-ld

# The magic debhelper rule:
%:
        dh --with quilt $@

override_dh_auto_configure:
        # remove out of date files
        rm -f conf/config.guess conf/config.sub
        autoreconf -fi
        dh_auto_configure

build:
        dh build
        $(MAKE) docs

clean:
        dh clean
        rm -rf docs
```

これもスゴいなぁと。 下手したら、 CDBS より覚えやすいんじゃないだろうか? とか。

参考文献

[Debian Policy Manual] Ian Jackson & Christian Schwarz, 1996: Debian Policy Manual, http://www.debian.org/doc/debian-policy/

[やまだ & 鵜飼 (2006)] やまだあきら (著), 鵜飼文敏 (監修), 2006: 入門 Debian パッケージ, 技術評論社, ISBN4-7741-2768-X

[CDBS Documentation Rev. 0.4.0] Marc (Duck) Dequénes, Arnaud (Rtp) Patard, 2007: CDBS Documentation, http://perso.duckcorp.org/duck/cdbs-doc/cdbs-doc.xhtml

[CDBS Documentation Rev.0.1.2] Marc (Duck) Dequénes, Arnaud (Rtp) Patard, Peter Eisentraut, Colin Walters, 2007: /usr/share/doc/cdbs/cdbs-doc.html.

[Online CDBS Gallery] Online CDBS Gallery, http://cdbs.ueberalles.net/index.html

[Debian パッケージ作成の手引き] 小林 儀匡, Debian パッケージ作成の手引き, http://www.debian.or.jp/~nori/debian-packaging-guide/index.html

[CDBS 1st step] 佐々木洋平, 2008: はじめての CDBS, 第 18 回関西 Debian 勉強会 2008 年 10 月 配布資料 http://tokyodebian.alioth.debian.org/pdf/debianmeetingresume200810-kansai.pdf

[lintian] 大浦 真、 2009: 「lintian でパッケージをチェックする」、第 26 回関西 Debian 勉強会 2009 年 8 月 配布資料 http://tokyodebian.alioth.debian.org/pdf/debianmeetingresume200908-kansai.pdf

14 lintian でパッケージをチェックする

大浦 真

14.1 lintian とは

- Debian パッケージには、さまざまなポリシーや作法が存在し、主に以下の文書にまとまっています。
  - Debian ポリシーマニュアル (`http://www.debian.org/doc/debian-policy/`)
  - Debian 開発者リファレンス (`http://www.debian.org/doc/developers-reference/`)
- これらの内容を全て手動でチェックするのは困難なので、チェックするために **lintian** というツールが存在します。
- パッケージがポリシーに合致しているかどうかをある程度チェックできます。(完全に全てチェックできるわけではありません。)
- 元々 C 言語で文法をチェックするためのツールとして **lint** というものが存在しました。
  - lint とは、元々、機織りの際に出る糸くずやほつれ糸を指します。
  - Debian には、**splint** というパッケージが存在します。
  - C 言語以外でも文法チェックのツールの名前としてよく使われます。`http://openlab.ring.gr.jp/k16/htmllint/` (Another HTML-lint) など。
- lintian は 1998 年に開発が始まりました。perl を使ったツールです。
- 一時期、同様の機能を持った **linda** というツールもありましたが、こちらは lenny から削除さました。

14.2 lintian の使い方

- 完成したバイナリパッケージ (`.deb`) とソースパッケージ (`.dsc`) に対してチェックを行います。
- **debuild** でパッケージをビルドするとビルド後に自動的に `lintian` が実行されます。
- 手動でチェックする場合は、`lintian` コマンドの引数に`.changes` ファイルを指定すると、バイナリパッケージとソースパッケージを同時にチェックできます。(`.changes` の中には、バイナリパッケージとソースパッケージのファイル名が書かれています。)

  -c　通常のチェックを行います。(省略可)

  -i　詳しい説明も同時に表示します。

  -I　付加的な内容のチェックも行います。

14.3 具体例

14.3.1 hello-debhelper の場合

- 以下では、**hello-debhelper**(2.4-1) をサンプルとして説明します。(**lintian** のバージョンは 2.2.14。)

- オプションなしで .changes ファイルを指定して lintian を実行します。

```
% lintian hello-debhelper_2.4-1_amd64.changes
E: hello-debhelper: package-contains-info-dir-file usr/share/info/dir.gz
W: hello-debhelper: install-info-used-in-maintainer-script prerm:3
W: hello-debhelper: install-info-used-in-maintainer-script postinst:4
W: hello-debhelper: missing-dependency-on-install-info
```

- 'E: ' から始まるものはエラー、'W: ' から始まるものは警告を意味します。
- 詳しい内容は -i オプションで見ることができます。

```
% lintian -i hello-debhelper_2.4-1_amd64.changes
E: hello-debhelper: package-contains-info-dir-file usr/share/info/dir.gz
N:
N:    This package contains a file named dir or dir.old, possibly compressed,
N:    in /usr/share/info. This is the directory (or backup) of info pages and
N:    is generated automatically by install-info when a package containing
N:    info documentation is installed. Some upstream build systems create it
N:    automatically, but it must not be included in a package since it needs
N:    to be generated dynamically based on the installed info files on the
N:    system.
N:
N:    Severity: important, Certainty: certain
N:
W: hello-debhelper: install-info-used-in-maintainer-script prerm:3
N:
N:    This script apparently runs install-info. Updating the
N:    /usr/share/info/dir file is now handled automatically by triggers, so
N:    running install-info from maintainer scripts is no longer necessary.
N:
N:    If debhelper generated the maintainer script fragment, rebuilding the
N:    package with debhelper 7.2.17 or later will fix this problem.
N:
N:    Severity: normal, Certainty: possible
N:
W: hello-debhelper: install-info-used-in-maintainer-script postinst:4
W: hello-debhelper: missing-dependency-on-install-info
N:
N:    This package appears to contain at least one info document but does not
N:    depend on dpkg (>= 1.15.4) | install-info as recommended by Policy. This
N:    dependency is needed for the transition to triggerized install-info to
N:    correctly build the info directory during partial upgrades from lenny.
N:
N:    Refer to Debian Policy Manual section 12.2 (Info documents) for details.
N:
N:    Severity: normal, Certainty: possible
N:
```

- 一つ目のエラー: 「パッケージの中に usr/share/info/dir.gz が含まれている。」
  - info ディレクトリの中の dir ファイルは info の目次に相当するもので、Debian では、install-info コマンドで自動的に作られるものなので、パッケージに含めるべきではありません。
  - debian/rules の install ターゲットの中で dir ファイルを削除します。
- 二つ目、三つ目の警告: 「install-info コマンドが postint と prerm の中で実行されている。」
  - install-info コマンドは、dpkg の trigger という機能で自動的に実行されるので、postinst と prerm で実行する必要はありません。
  - debian/postinst と debian/prerm を削除。(このパッケージでは、他に postinst、prerm を使っていないので、install-info コマンドの部分だけを消すと、「postinst、prerm が空っぽだ」という警告が出ます。)
- 四つ目の警告: 「install-info への依存がない。」
  - install-info コマンドが入っている **install-info** パッケージに対する依存関係がありません。Debian ポリシー 12.2 参照。

    http://www.debian.org/doc/debian-policy/ch-docs.html#s12.2
  - debian/control に 'dpkg (>= 1.15.4) | install-info' と追加します。
- これで 'lintian clean' になります。

14.3.2 よくある警告

- よく見かける警告

```
W: hello-debhelper source: out-of-date-standards-version 3.8.0 (current is 3.8.3)
W: hello-debhelper: binary-without-manpage usr/bin/hello2
```

```
W: hello-debhelper source: out-of-date-standards-version 3.8.0 (current is 3.8.3)
N:
N:    The source package refers to a Standards-Version older than the one that
N:    was current at the time the package was created (according to the
N:    timestamp of the latest debian/changelog entry). Please consider
N:    updating the package to current Policy and setting this control field
N:    appropriately.
N:
N:    If the package is already compliant with the current standards, you
N:    don't have to re-upload the package just to adjust the Standards-Version
N:    control field. However, please remember to update this field next time
N:    you upload the package.
N:
N:    See /usr/share/doc/debian-policy/upgrading-checklist.txt.gz in the
N:    debian-policy package for a summary of changes in newer versions of
N:    Policy.
N:
N:    Severity: normal, Certainty: certain
N:
W: hello-debhelper: binary-without-manpage usr/bin/hello2
N:
N:    Each binary in /usr/bin, /usr/sbin, /bin, /sbin or /usr/games should
N:    have a manual page
N:
N:    Note that though the man program has the capability to check for several
N:    program names in the NAMES section, each of these programs should have
N:    its own manual page (a symbolic link to the appropriate manual page is
N:    sufficient) because other manual page viewers such as xman or tkman
N:    don't support this.
N:
N:    If the name of the man page differs from the binary by case, man may be
N:    able to find it anyway; however, it is still best practice to make the
N:    case of the man page match the case of the binary.
N:
N:    If the man pages are provided by another package on which this package
N:    depends, lintian may not be able to determine that man pages are
N:    available. In this case, after confirming that all binaries do have man
N:    pages after this package and its dependencies are installed, please add
N:    a lintian override.
N:
N:    Refer to Debian Policy Manual section 12.1 (Manual pages) for details.
N:
N:    Severity: normal, Certainty: possible
```

- 一つ目の警告: 「Standards-Version が古い。」
  - Debian ポリシー、および、それに付随する文書のバージョンを Standards-Version と呼びます。debian-policy パッケージのバージョン番号で、現時点での最新版は 3.8.3.0 です。パッケージは、どの Standards-Version に従っているか debian/control で指定しますが、古すぎると警告が出ます。
  - debian-policy パッケージの中の upgrading-checklist.txt で変更点を確認して、パッケージを更新します。
- 二つ目の警告: 「マニュアルページがない。」
  - /usr/bin、/usr/sbin、/bin、/sbin などにあるバイナリに対してマニュアルページがありません。Debian としては、コマンドに対するマニュアルページは必須になります。Debian ポリシー 12.1 参照。
  - 上流のソースにマニュアルページがあればそれをインストールし、なければ自分で作成します。

14.4 まとめ

- lintian はスクリプトでパッケージをチェックするだけなので、ポリシーの全ての内容を確認することはできません。必要に応じてポリシーを自分で確認する必要があります。
- 事情により、どうしてもチェックを外したい時は、overrides を使うことができます。(/usr/share/lintian/overrides/ 参照)
  - /usr/share/lintian/overrides/adduser の内容

```
adduser: maintainer-script-needs-depends-on-adduser postinst
adduser: maintainer-script-needs-depends-on-adduser postrm
adduser: maintainer-script-needs-depends-on-adduser config
```

- チェックの内容を調べることにより、パッケージングシステムの仕組みを深く知ることができます。

15 Debian Mentors ってご存知ですか

佐々木洋平

15.1 はじめに

これまでの勉強会で「Debian パッケージの作成方法」について色々学んできました。作成したパッケージを個人的用途の為にだけ使用するなら「パッケージができた!!」でオシマイですが、どうせなら公式配布物に入れてみたくありませんか?

ここでは Debian Developer じゃない人や Debian Developer を目指している人の第一歩として「Package Maintainer」になるまでについて書いてみます。

15.2 Package Maintainer?

公式配布されているパッケージを作成/更新している人々は三つに分類されます:

Debian Developer(以下、 DD) 公式開発者。 **偉い人**。

Debian Maintainer(以下、 DM) パッケージを沢山作成している人。公式開発者ではないが、作成したパッケージを公式配布物へ upload する権限を持っている人。

Package Maintainer(以下、 PM) パッケージを作成している人。upload の権限は無い。にスポンサーになってもらって upload してもらう人。

パッケージを公式配布物とするための第一歩は PM になることです。そのためにはスポンサーになってくださる DD を探さなければなりません。

15.3 Debian Mentros 制度

運良くスポンサーを引き受けて下さる Debian Developer がみつかったら、 その人の指導のもとで (修正点があれば) パッケージの修正を行ない、 公式配布物へスポンサー upload してもらうことになります。 スポンサーを引き受けて下さった DD は、 PM にとって **mentor** にあたります。mentor は「賢明で信頼のおける相談相手 (顧問)、 指導者、 師、 先生」という意味です。

佐々木は幸いな事に岩松さんにスポンサーを引き受けていただきました。岩松さんの指導のもと、 先日ようやく一つ目のパッケージが公式に入りました。

```
$ apt-cache show rttool
Package: rttool
Priority: extra
Section: text
Installed-Size: 148
Maintainer: Youhei SASAKI <uwabami@gfd-dennou.org>
Architecture: all
Version: 1.0.3-1
Depends: librt-ruby1.8 (= 1.0.3-1), ruby1.8
Filename: pool/main/r/rttool/rttool_1.0.3-1_all.deb
Size: 15720
MD5sum: a232c92d41a80339331e05d4243715cc
SHA1: 549b1826892e327bbafac982eb1e6397a9f2965f
SHA256: d134673d709983d0cbb67d8c7ee8180c6b4dce25f480299874de62a30b0f6a05
Description: RT table formatter
 RT is simple human-readble table format. RTtool is a converter form RT into
 various formats. RTtool is one of frontends of formatter for RT.
 .
 This package provides rt2 command.
Homepage: http://www.rubyist.net/~rubikitch/computer/rttool/index.en.html
```

「千里の道も一歩から」とか「小さな事からコツコツと (©西川きよし師匠)」と言います。 DD への第一歩、 だと良いなぁ。。。

15.4 mentors.debian.net

では実際にパッケージを作成したのでスポンサーを探してみましょう。スポンサーを探す際には、作成したパッケージのソースファイル一式をスポンサーが取得できる場所に置き、スポンサー募集のメールを書いたりします。この手続きをサポートするサイトが mentors.debian.net です。

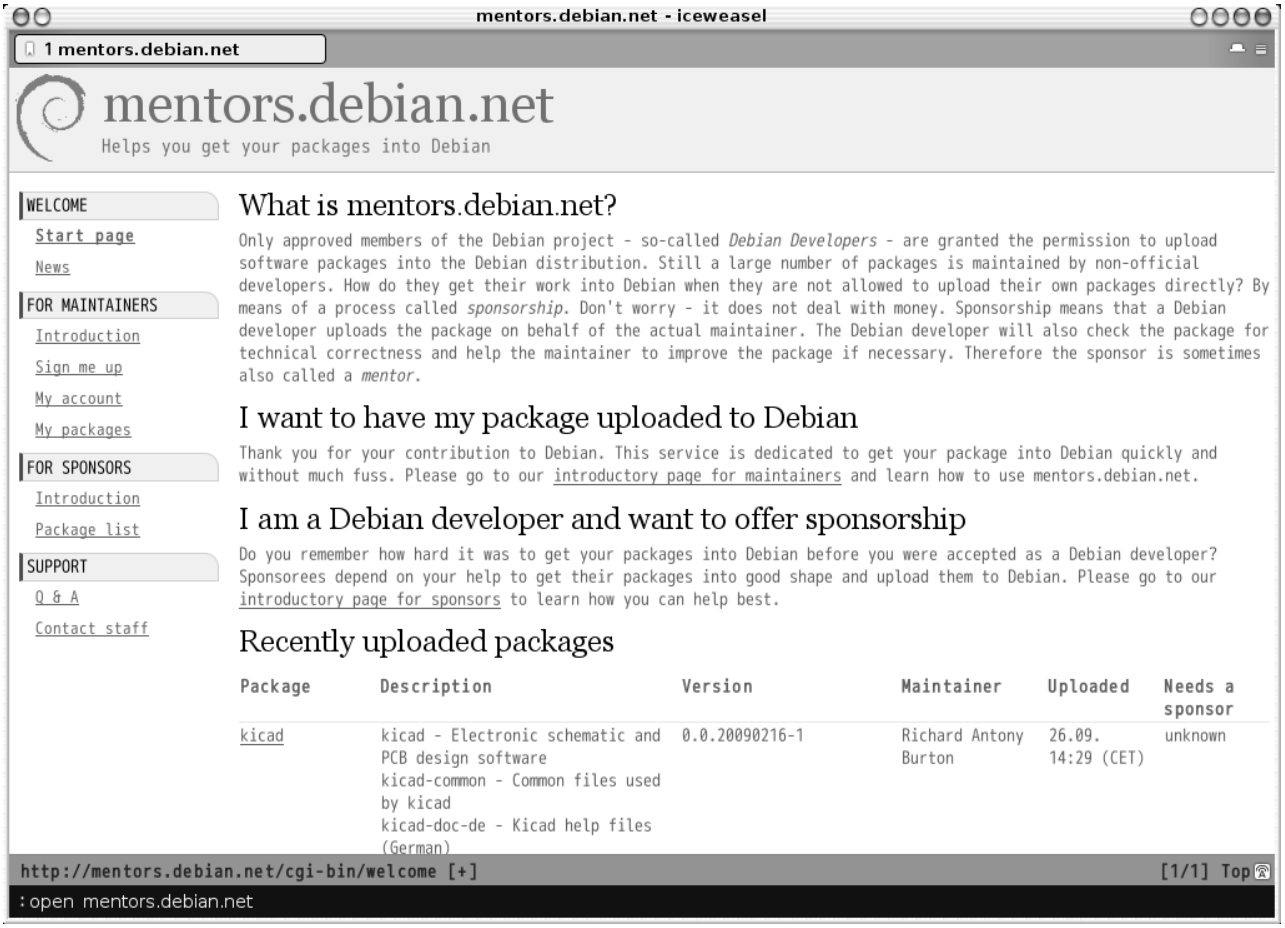


図 15.1 http://mentors.debian.net

サイト左側のメニューにある「FOR MAINTAINERS」の項目にある「Introduction」を見てみて下さい。PM が

作成したパッケージが公式配布物に含まれるまでの流れが書いてあります。また、 このページの下の方には dput や dupload を使用してソース一式を upload する際の設定例が載っています。良く読んでおきましょう。

次に「sign up」からアカウントを取得します。

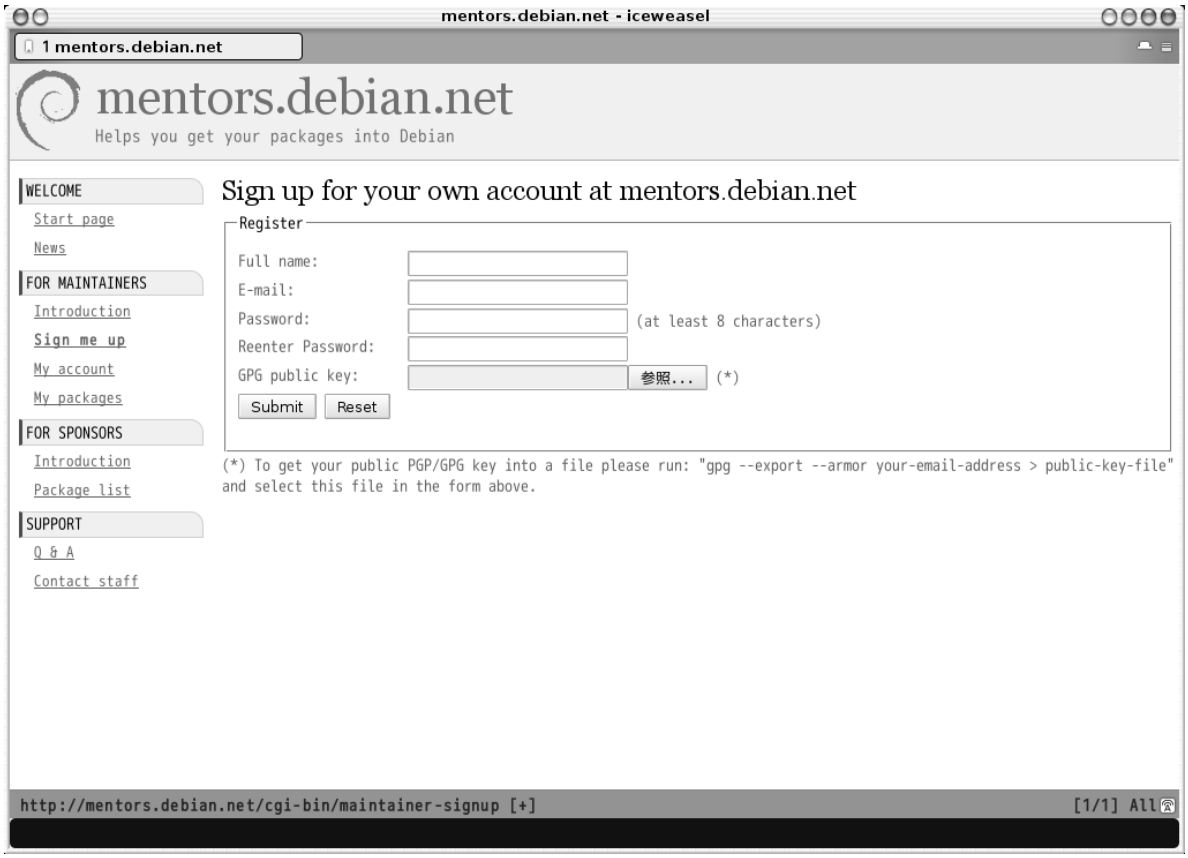


図 15.2　http://mentors.debian.net/cgi-bin/maintainer-signup

アカウントの取得には GPG 公開鍵が必要になりますこの公開鍵はパッケージ作成時の署名に用いた GPG 鍵を使用して下さい。

では実際にアカウントを取得して mentors.debian.net を使用してみましょう。

おおっと!

スポンサーを探す前に以下の事は済んでいますか?

1. ITP bug report
2. 作成するパッケージを lintian clean にする

一つ目は「これからこのソフトウェアのパッケージを作成するぞ!」という宣言ですね。WNPP – ITP としてバグレポートを書きます。WNPP は「Work-Needing and Prospective Packages」の略です

Work-Needing and Prospective Packages: WNPP
http://www.debian.org/devel/wnpp

バグレポート書き方についてはのがたさんの文章なんかを参考にして下さい。

二つ目はパッケージ作成の間違いの修正です。lintian については前回の大浦さんの発表 [1] なんかを参考にして下さい。

さあ準備はできましたか?

15.4.1 sign up!

アカウント取得のプロセス自体は非常に単純です。 名前、 メールアドレス、 パスワード、 GPG 公開鍵を入力してしばし待ちましょう。 登録したメールアドレスに confirm メールが来ますので、 そこに書かれた URL をブラウザで開いて下さい。 アカウントが有効になります。

15.4.2 dupload の設定

ソースファイル一式のアップロードには dput もしくは dupload を使用します。先ず dupload パッケージを導入しましょう

```
$ sudo aptitude install dupload
```

dupload の設定ファイルは `/etc/dupload.conf` もしくは `~/.dupload.conf` です. dupload パッケージの提供する `/etc/dupload.conf` には mentors.debian.net の設定がすでにあります. もし無い場合には以下の内容を `~/.dupload.conf` に記述して下さい。

```
package cofig;
$cfg{'mentors'} =
{
  fqdn => 'mentors.debian.net',
  incoming => '/',
  dinstall_runs => 1,
  passive => 1,
};

1;
```

15.4.3 upload!

さて、 ここまで終わったら upload してみましょう. 以下では rttool というパッケージを upload してみます.

```
$ dupload -t mentors  rttool_1.0.3-2_amd64.changes
dupload note: no announcement will be sent.
Checking signatures before upload...GPG signature is missing
dupload fatal error: Pre-upload '/usr/share/dupload/gpg-check %1'
failed for rttool_1.0.3-2_amd64.changes at /usr/bin/dupload line 223
```

... 怒られてしまいました。

お怒りの原因は明確ですね。 パッケージ作成時に GPG 署名をしていないからです。この様に dupload には `.dsc` や `.changes` が GPG 署名されているのか、などの検証も行なってくれます。

ではちゃんと署名してパッケージを作成した後に再度 upload してみましょう。

```
dupload -t mentors rttool_1.0.3-2_amd64.changes
dupload note: no announcement will be sent.
Checking signatures before upload......signatures are ok
Uploading (ftp) to mentors.debian.net:/
[ job rttool_1.0.3-2_amd64 from rttool_1.0.3-2_amd64.changes
 rttool_1.0.3-2_all.deb, size ok, md5sum ok, sha1sum ok, sha256sum ok
 rttool_1.0.3-2.dsc, size ok, md5sum ok, sha1sum ok, sha256sum ok
 librt-ruby1.8_1.0.3-2_all.deb, size ok, md5sum ok, sha1sum ok, sha256sum ok
 rttool_1.0.3-2.diff.gz, size ok, md5sum ok, sha1sum ok, sha256sum ok
 rttool_1.0.3.orig.tar.gz, size ok, md5sum ok, sha1sum ok, sha256sum ok
 rttool_1.0.3-2_amd64.changes ok ]
Uploading (ftp) to mentors (mentors.debian.net)
+ FTP passive mode selected
[ Uploading job rttool_1.0.3-2_amd64
 rttool_1.0.3-2_all.deb 15.1 kB, ok (1 s, 15.07 kB/s)
 rttool_1.0.3-2.dsc 1.1 kB, ok (1 s, 1.06 kB/s)
 librt-ruby1.8_1.0.3-2_all.deb 15.2 kB, ok (2 s, 7.59 kB/s)
 rttool_1.0.3-2.diff.gz 2.7 kB, ok (1 s, 2.68 kB/s)
 rttool_1.0.3.orig.tar.gz 35.0 kB, ok (2 s, 17.48 kB/s)
 rttool_1.0.3-2_amd64.changes 1.9 kB, ok (3 s, 0.65 kB/s) ]
```

無事 upload された様です。この後「upload されたよ」というメールが届きます。

```
Subject: 'rttool' uploaded to mentors.debian.net
From: ``mentors.debian.net'' <support@mentors.debian.net>
To: uwabami@gfd-dennou.org
Date: Sat, 26 Sep 2009 16:33:47 +0200 (CEST)

Your upload of the package 'rttool' to mentors.debian.net was
successful. Sponsors can now download it. The URL of your package is:
http://mentors.debian.net/debian/pool/main/r/rttool

The respective dsc file can be found at:
http://mentors.debian.net/debian/pool/main/r/rttool/rttool_1.0.3-2.dsc
-----
```

mentors.debian.net に login してパッケージの情報を見てみましょう。

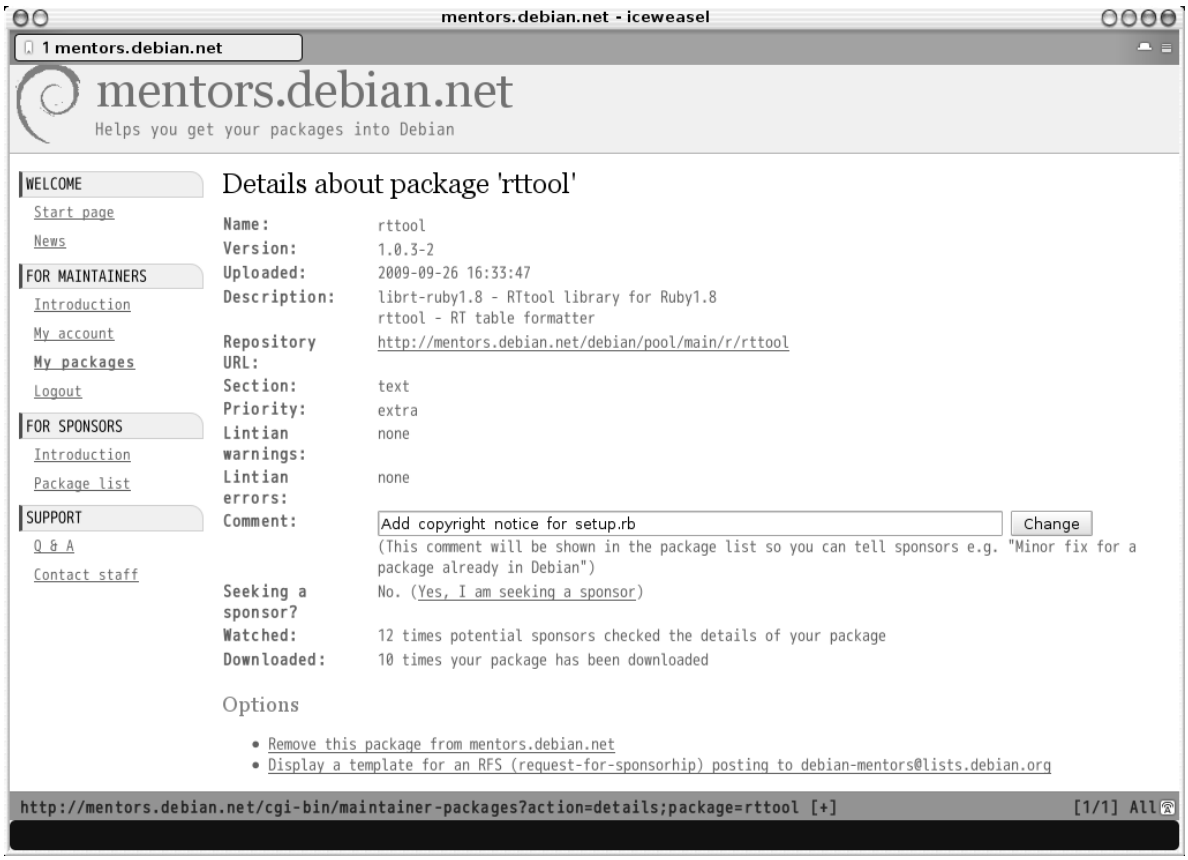


図 15.3　http://mentors.debian.net/cgi-bin/maintainer-packages?action=details;package=rttool

この時点で lintian warning & error が無いか確認しておきましょう。unstable 環境じゃないと lintian が古くて check しきれていない場合もあります。

また、 スポンサー募集中ならば「Seeking a sponsor?」の項目を 「Yes」 にしておくと良いと思います。

ここまできたら準備は完了です。 スポンサーを募ってみましょう。debian-mentors@lists.debian.org や debian-devel@debian.or.jp などでスポンサー募集のメールを送信してみます。関西 Debian 勉強会参加者でしたら、 木下さんや大浦さんに伺ってみるのも良いかもしれません。 顔がわかっている人同士の方が話が進みやすいかもしれません。

「debian-mentors@lists.debian.org は英語なのでちょっと...」という場合もご心配無く。mentors.debian.net に upload したパッケージ情報の所を見て下さい。 debian-mentors にメールを送るための雛形があります。 これを元にしてメールを書いてみると良いでしょう。

15.5　まとめ

Debian の Mentor 制度と mentors.debian.net について紹介しました。実際には mentor がみつかるかどうかは定かではありませんので、 道程は長いです。

また、 今回の文章には DD としてスポンサーする側の情報がありません。Mentor になる側として「こういう感じだとスポンサーになりやすい/なってやろうかという気になる」というご意見があったら聞いてみたい、 と思いました。

参考文献

[1] 大浦 真、 2009: 「lintian でパッケージをチェックする」、第 26 回関西 Debian 勉強会 2009 年 8 月 配布資料 http://tokyodebian.alioth.debian.org/pdf/debianmeetingresume200908-kansai.pdf

16 Debian Autobuilder ネットワーク

岩松 信洋

Debian は現時点で 11 個の CPU アーキテクチャと 2 つのカーネルをサポートしています (i386, amd64, ppc, mips, mipsel, armel, alpha, ia64, s390, hppa, sparc, kfreebsd-amd64, kfreebsd-i386)。

パッケージが数万個あり、パッケージメンテナが Debian のサーバにアップロードすると、いつのまにかユーザが知らないうちに各アーキテクチャ向けに Debian パッケージがビルドされ、利用できるようになっています。これはどのような仕組みで動いているのでしょうか。もちろん、パッケージメンテナはすべてのアーキテクチャを持っているわけではないので（持っている人もいるみたいですが。）、パッケージメンテナが各アーキテクチャ毎にビルドして、アップロードしてわけではありません。Debian では **Autobuilder ネットワーク** と呼ばれる 各アーキテクチャ向けにパッケージが自動的にビルドされるシステムが行っています。Debian Project では重要なシステムの一つなのですが、表に出てくる事も少ないためどのような動きになっているのか知る人は少ないです。今回、Renesas SuperH(SH4) を移植するにあたりこれらの情報を入手したのでまとめてみました。また、新しくアーキテクチャを追加する時の方法と、ビルドテクニックについても説明します。

16.1 Autobuilder ネットワークのコンポーネント

Autobuilder ネットワーク は一つのソフトウェアで動いているわけではなく、いくつかのソフトウェアと porter と呼ばれる buildd のメンテナ、wanna-build のメンテナで構成されています。Autobuilder ネットワークの動きを説明する前に、各コンポーネントがどのような動きをするのか説明します。

16.1.1 dak

Debian Archive Kit。パッケージメンテナ/Buildd によってアップロードされたパッケージをチェック、管理するためのツールやデーモンが提供されています。

16.1.2 quinn-diff

パッケージメンテナによって新しくアップロードされたソフトウェアのアーキテクチャ依存のパッケージ (Architecture: any) とアーキテクチャ非依存のパッケージ (Architecture: all) をチェックし、前者を wanna-build のデータベースに登録するツールです。これはソースパッケージだけでなく、バイナリパッケージもチェックします。よって、Autobuilder ネットワークでは アーキテクチャ非依存のパッケージはビルドされません。例えば、dpkg ソースパッケージからは dpkg, dpkg-dev, dselect バイナリパッケージが生成されますが、dpkg-dev はアーキテクチャ非依存のパッケージなので、Autobuilder ではビルドされません。アーキテクチャ毎にビルドしてしまうとパッケージのハッシュ値が変わってしまうためです。これはパッケージメンテナがアップロードした dpkg-dev パッケージを利用することになります。

パッケージ情報: `http://packages.qa.debian.org/q/quinn-diff.html`

16.1.3 wanna-build

buildd とデータのやりとりをするためのインターフェイスです。各アーキテクチャ毎にパッケージのデータベースを持ちます。buildd は wanna-build のデータベースを持つサーバに ssh でログインし、wanna-build コマンドを使ってデータベースをコントロールします。

パッケージ情報: `http://packages.debian.org/sid/wanna-build`

16.1.4 buildd

wanna-build と データをやりとりするデーモンやパッケージをアップロードするツール一式を指します。ツールには以下のものがあります。

パッケージ情報: `http://packages.debian.org/sid/buildd`

- buildd
  buildd デーモン。wanna-buildd のデータベースを定期的にチェックし、ビルドが必要なパッケージをパッケージビルドキューとして管理します。そして、sbuild へのビルド指示を行います。
- buildd-mail
  メールキューディレクトリにあるデータを処理するプログラム。ビルドに成功したパッケージ (*.deb, *.udeb) とパッケージ情報 (*.dsc, *.changes, etc) をアップロードキューディレクトリに移動させます。
- buildd-mail-wrapper
  buildd-mail-wrapper は porter から送られたメールをメールキューディレクトリに格納し、buildd-mail を呼び出すプログラムです。入力データとして、メールデータそのものを要求するので、procmail など合わせて利用します。
- buildd-update-chroots
  パッケージのビルドを行う chroot 環境をアップデートおよびクリーンアップするプログラムです。buildd が動作している場合には buildd を停止し、chroot をアップーデートします。アップデート完了後、再度 buildd を起動させます。クリーンアップには debfoster と呼ばれるゴミパッケージを消すプログラムが利用されています。
- buildd-uploader
  アップロードキューディレクトリにあるパッケージをアップロードするプログラムです。dupload を使い、GPG サインされている *.changes ファイルをパーサし、パッケージをアップロードします。cron 等で定期的に呼び出して利用します。
- buildd-watcher
  buildd の動きをチェックするプログラムです。cron 等で定期的に呼び出して利用します。

正式な buildd では https://buildd.debian.org/apt/pool/からダウンロードしたパッケージを使う必要があります。互換性維持のため、手を加えた sbuild を使う必要あるためです。

16.1.5 sbuild

実際にパッケージをビルドするプログラムです。buildd から呼ばれます。パッケージのソースコードを取得し、ビルド結果を porter と wanna-build に送信するまでの処理を行います。pbuilder と機能がかぶりますが、sbuild はアーキテクチャ依存部分のビルドに特化したパッケージビルドシステムになっています。schroot を利用することにより、lvm などディスク管理機能にも対応もされています。

パッケージ情報: `http://packages.debian.org/sid/sbuild`

正式な buildd では `https://buildd.debian.org/apt/pool/` からダウンロードしたパッケージを使う必要があります。互換性維持のため、手を加えた sbuild を使う必要あるためです。

16.1.6 schroot

sbuild から呼ばれる chroot 代替プログラムです。chroot の高機能版です。

パッケージ情報: `http://packages.debian.org/sid/schroot`

16.1.7 porter

buildd をメンテナンスする人です。ビルド結果をメールとして受信し、その結果を buildd に対して返信します。ビルドに成功したパッケージのメールには GPG でサインして返信します。返信する際には、送られてきたメールから、*.changelog の部分を抜き出して、その抜き出した部分にサインを行います。抜きだしの部分にはスクリプトを使って処理するのですが、現在のところ mutt のスクリプトしかないので、ほとんどの porter は Mutt ユーザです。また、GPG によるサインを自動化すればよいのですが、セキュリティの観点から porter がサインをする仕組みを取っています。

ビルドに失敗した場合のメールの書式も決まっています。以下に書式を示します。

- ビルドに失敗した場合

```
failed
理由（BTS の番号など）
```

- パッケージの依存関係の問題

```
dep-wait 依存するパッケージの名前とバージョン
```

- 再ビルド申請

```
give-back
```

- パッケージがアーキテクチャ依存の場合

```
not-for-us
```

*30

16.1.8 wanna-build メンテナ

メールサーバ障害や、buildd の障害でうまくパッケージがビルドできない事があります。この場合には wanna-build のデータベースにアクセスし、パッケージのステータスを元に戻す作業が必要になる場合があります。この場合、porter が wanna-build のデータベースにアクセスして変更するわけではなく、wanna-build メンテナが依頼を受けて、変更処理を行います。

16.2 Autobuilder ネットワークの動き

さて、Autobuilder ネットワークのコンポーネントの説明をしたので、次に各コンポーネントの連携について説明します。先で説明した各コンポーネントの連携図を図 16.1 に示します。

16.3 パッケージの状態遷移

パッケージは常になにかしらの状態をステータスを持っています。ステータスは wanna-build で管理する データベース で保持されています。ステータスは以下のものがあり、図 16.2 のように変化します。

*30 アーキテクチャリストに入っていないだけの場合は、failed を指定し、ビルドエラーにした後、ポーティングを行い、パッチを BTS します。

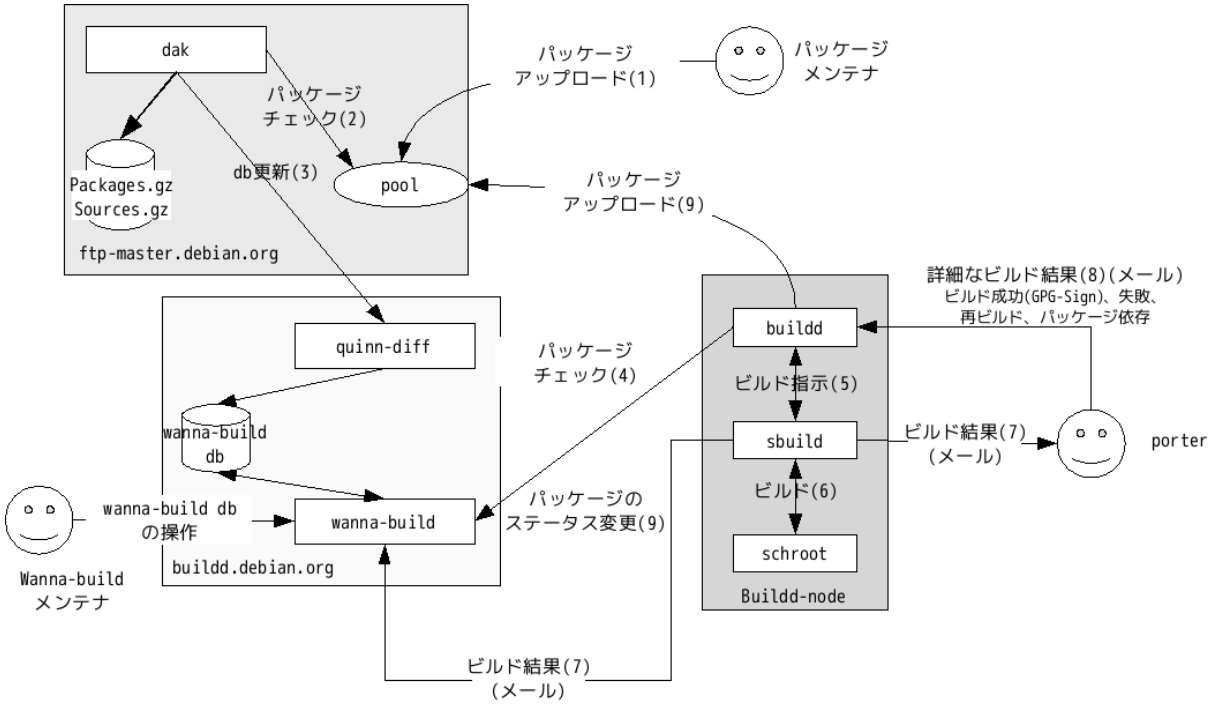


図 16.1　コンポーネント連系図

- BD-Uninstallable
  パッケージのビルドに足りないパッケージがある状態。
- Needs-Build
  パッケージがビルド可能な状態。パッケージビルド依存関係が解消されてたり、ビルドの状態から戻された (given-back) 場合にこの状態になります。
- Building
  Buildd によってパッケージがビルドされている状態。
- Uploaded
  Porter によってパッケージに GPG サインされ、アーカイブにアップロードされた状態。
- Installed
  アーカイブにインストールされた状態。
- Dep-Wait
  パッケージのビルド依存関係待ちの状態。
- Failed
  パッケージがビルドに失敗した状態。
- Not-For-Us
  対象のアーキテクチャではビルドできないパッケージの状態。porter が wanna-build 経由でデータベースに登録します。現在、wanna-build のデータベースとは別にアーキテクチャ依存のリスト[*31]を作成中です。
- Failed-Removed
  Failed 状態のパッケージが一時的に削除されているという状態。

[*31] http://buildd.debian.org/git/packages-arch-specific.git

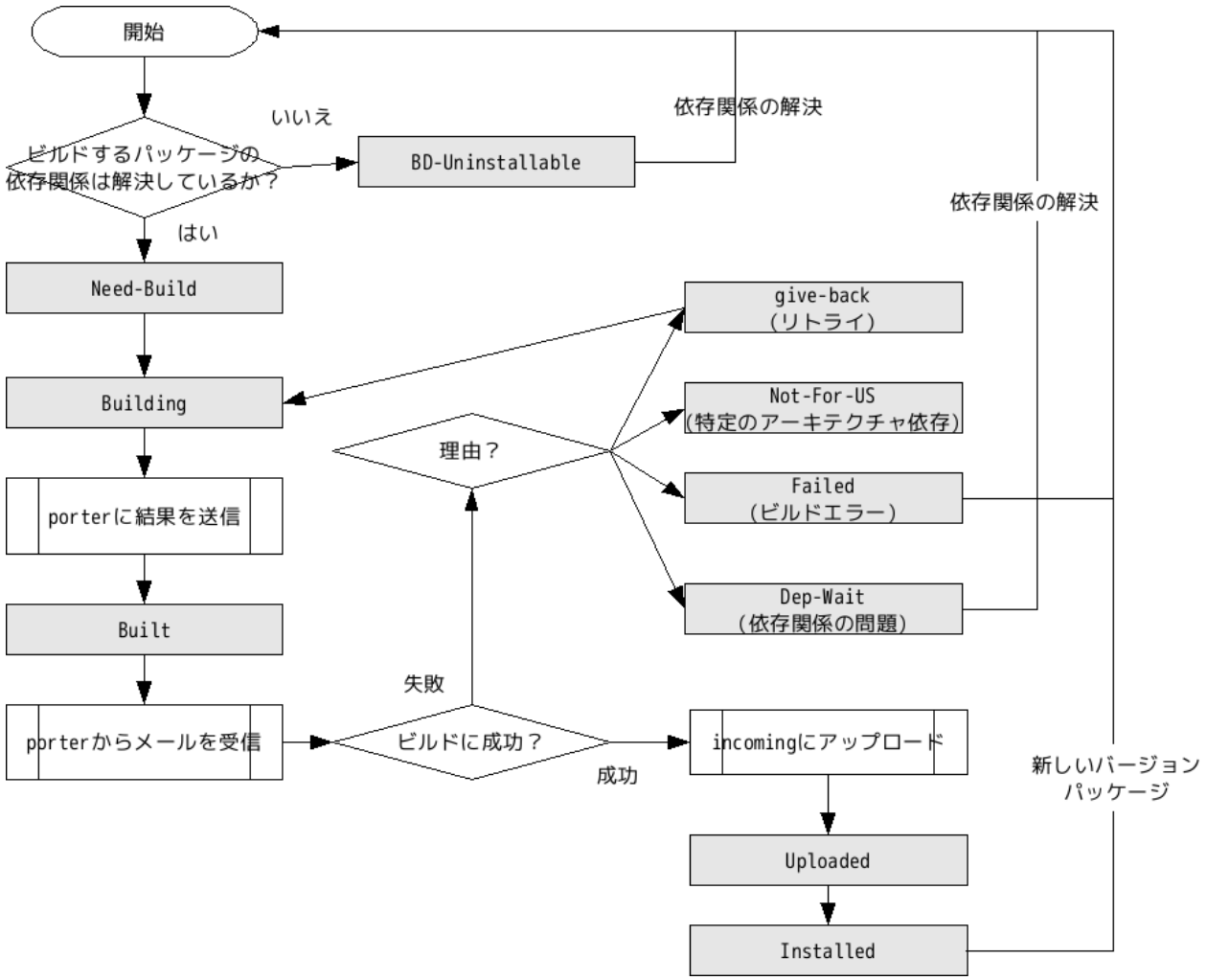


図 16.2　パッケージステータスの状態遷移

16.4　ビルドの優先順位

パッケージのビルドは基本的に Need-Build のステータスになった順かつパッケージ名 ASCII 順でビルドされますが、それとは別にソースパッケージのプライオリティとソースパッケージのセクション毎に優先順位が設定されています。現在では、以下の優先順位になっています。

16.5　新しいアーキテクチャ/カーネルをサポートする

昔は、ポーティングしたい人が quinn-diff から buildd(buildd-node) まで全て構築する必要がありました。この最大のネックは wanna-build データベースで、これは最新のものを使う必要があったのですが、データベースにアクセスするにはサーバーのアカウントが必要だったため、構築が困難でした[*32]。しかし、今は新しいアーキテクチャを追加するためのサポート用サービス debian-ports.org が用意されています。このサービスでは、ポーティングしたいアーキテクチャ用の wanna-build 用アカウント と パッケージ用データベース, quinn-diff のサービスを提供できるようになっています。これにより、移植したい人は buildd-node のみを用意し、debian-ports の管理者に登録申請をするだけで新しいアーキテクチャをサポートできるようになりました。現在、表 16.4 のアーキテクチャが

[*32] 構築は可能ですが、タイムラグが発生します。その理由として buildd が参照するリポジトリは最新のリポジトリ情報が反映される buildd 専用のためです

Source Priority	値
required	-5
important	-4
standard	-3
optional	-2
extra	1
unknown	-1

表 16.1　ソースパッケージプライオリティ毎の優先順位

libs	-200
debian-installer	-199
base	-198
devel	-197
shells	-196
perl	-195
python	-194
graphics	-193
admin	-192
utils	-191
x11	-190
editors	-189
net	-188
mail	-187
news	-186
tex	-185
text	-184

表 16.2　ソースパッケージセクション毎の優先順位:その 1

web	-183
doc	-182
interpreters	-181
gnome	-180
kde	-179
games	-178
misc	-177
otherosfs	-176
oldlibs	-175
libdevel	-174
sound	-173
math	-172
science	-171
comm	-170
electronics	-169
hamradio	-168
embedded	-166
unknown	-165

表 16.3　ソースパッケージセクション毎の優先順位:その 2

アーキテクチャ名	進捗
avr32	65.52%
hurd-i386	64.46%
m68k	53.00%
sh4	85.43%

表 16.4　debian-ports.org でサポート中のアーキテクチャ

debian-ports.org を使ってポーティングを進めています。そして、kFreeBSD-amd64/i386 はこのサービスから誕生した最初のサポートアーキテクチャ(カーネル) になりました。

また、debian-ports.org では wanna-build のメンテナが存在しません。porter が wanna-build のメンテナンスをすることができます。正式にサポートされたアーキテクチャではないのでこのあたりはあまり厳密に決まっていないようです。

16.6　loop-depends の対応方法

いくつかのパッケージには loop-depends になっているものがあります。loop-depends とは、ビルド依存がループしているパッケージの状態を言います。例えば、libpang-perl パッケージ (図 16.3) は libgtk2-perl パッケージ図 16.4 にビルド依存しています。しかし、libgtk2-perl は libpang-perl パッケージに依存しています。

これではいつになってもビルドできません。これは新しいアーキテクチャをサポートする場合によく出てくる問題です。これはバグとして扱われるのでは？と思いますが、libpang-perl をビルドする段階で libgtk2-perl を使ったテストを行うため、libgtk2-perl が必要なのです。Debian ではソフトウェアのビルドに実行されるテストは行うことが推奨されているので、このようにビルド依存関係がループしています。では、どのようにしてビルドすればよいので

```
Source: libpango-perl
Section: perl
Priority: optional
Build-Depends: debhelper (>= 7.0.50), libcairo-perl (>= 1.000), libextutils-depends-perl (>= 0.300),
  libextutils-pkgconfig-perl, libglib-perl (>= 1:1.220), perl, xvfb, libpango1.0-dev,
  xauth, xfonts-base, quilt, libgtk2-perl (>= 1:1.220)
                             ^^^^^^^^^^^^^^^^^^^^^^^^^^
```

図 16.3　libpango-perl のビルド依存

```
Source: libgtk2-perl
Section: perl
Priority: optional
Build-Depends: debhelper (>= 7.0.50), quilt (>= 0.46-7), perl, libextutils-depends-perl (>= 0.300),
 libextutils-pkgconfig-perl (>= 1.03), libgtk2.0-dev (>= 2.6.0), libglib-perl (>= 1:1.220),
 libcairo-perl (>= 1.00), xvfb, xauth, xfonts-base, hicolor-icon-theme,
 libpango-perl (>= 1.220), shared-mime-info
 ^^^^^^^^^^^^^^^^^^^^^^^^^
```

図 16.4　libgtk2-perl のビルド依存

しょうか。実は、porter が手を加えてビルドする必要があります。しかし、手を加えた場合は正式なパッケージとして扱われません。このようなパッケージに遭遇した場合には、unreleased ディストリビューションにパッケージをアップロードし、そのパッケージを使って再度ビルドを行うという方法が推奨されています。

以下に例として libpang-perl をビルドする場合の手順を説明します。

1. buildd が動作していない別マシンでの作業
   (a) libpango-perl のソースパッケージをダウンロードします。
   (b) debian/control ファイルの Build-depends から libgtk2-perl を削除します。
   (c) debian/changelog をアップデートします。
   Debian version は既存のバージョンに +arch-name などをつける必要があります。(実際の Debian package と区別するため。) ディストリビューション名は unreleased に変更します。debian/changelog の変更者も変更する必要があります。
   (d) debuild -m"Your name <your-name@example.org>" などとしてパッケージをビルドします。
   libpango-perl は libgtk2-perl が見つからない場合には、テストを無視します。:-)
   (e) ビルドに成功したら、dupload を使ってパッケージアップロードします。
   (f) dak で処理されれば、unreleased ディストリビューションから libpango-perl が利用できるようになります。
2. schroot 内での作業
   (a) schroot 環境の apt-line に unreleased を追加します。
   (b) schroot 内で apt-get update を実行します。
3. libgtk2-perl がビルドできるようになります。
4. libgtk2-perl ができたら、unreleased ディストリビューションから libpango-perl を消してもらいます。
5. libpango-perl が 本来の Debian Version でビルドされます。

図にすると図 16.5 のようになります。

このループ依存はとても簡単な例なので、毎回この方法がうまくいくとは限りません。例えば、依存するパッケージや提供されるパッケージの機能によっては configure のオプションを変更する場合もあります (gtk2.0 や avahi など)。このあたりはパッケージメンテナか、debian-wb@l.d.o に直接聞いたほうがよいです。聞かないとわからないというのは問題なので、README.Debian あたりに書いてもらうように計画中です。

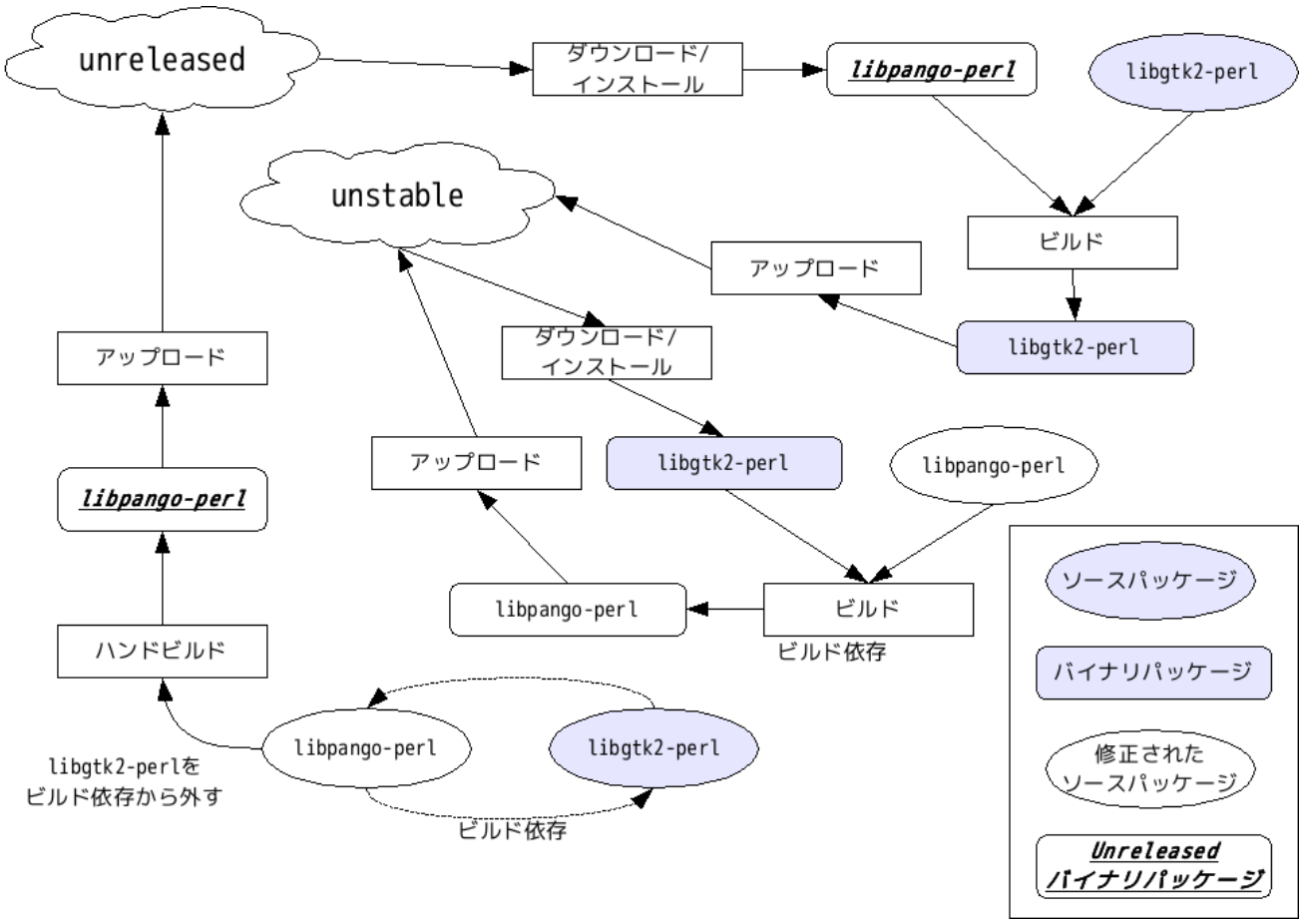


図 16.5　ループビルド依存関係回避方法

16.7　まとめ

Autobuilder ネットワークは Debian 公式開発者でさえあまり触ることのないシステムです。ブラックボックスなところが以前からあったと思うのですが、今回の説明で少しはわかる部分が増えたかなと思います。個人的には Auto と謳っておきながら実は人が介入していたりするのはけっこう意外でした。Autobuilder ネットワークの奥深くに入り込みたい人はぜひ Debian/SH4 のポーティングに参加していただければと思います。

17 Debian Trivia Quiz

上川 純一

ところで、みなさん Debian 関連の話題においついていますか？Debian 関連の話題はメーリングリストをよんでいると追跡できます。ただよんでいるだけでははりあいがないので、理解度のテストをします。特に一人だけでは意味がわからないところもあるかも知れません。みんなで一緒に読んでみましょう。

問題 1. ヨーロッパに新設されたアップロードキューの名前は?

☐ A ftp.eu.upload.debian.org

☐ B ftp.uk.upload.debian.org

☐ C ftp.jp.debian.org

問題 2. 新しいアップロードキューで新しくサポートすることになる通信プロトコルは?

☐ A ipv6

☐ B sstp

☐ C RFC2324

問題 3. GPG キー再作成祭りはなぜ発生したか?

☐ A そろそろ sha-1 が脆弱になったと思われるから

☐ B 惑星が直列するから

☐ C GPG が自由でなくなったから

問題 4. packages.debian.org メールについて何がアナウンスされたか

☐ A debian.org 以外からメールを受信しなくする

☐ B debian.org 以外からしかメールを受信しなくする

☐ C debian.net 以外からメールを受信しなくする

問題 5. eeePC は 5 月時点で何機種あるか

☐ A 16

☐ B 24

☐ C 32

問題 6. Debian が glibc の代わりに採用すると発表した libc はなにか

☐ A newlib

☐ B eglibc

☐ C BSD libc

問題 7. debian-cli というメーリングリストは何をするところか?
□ A command line interface について語る場所
□ B common language infrastructure について語る場所
□ C cat-linux interface について妄想する場所

問題 8. Debian policy 3.8.2 でかわった点はどれか。
□ A debconf 必須
□ B X は廃止になりました
□ C MS EULA が認定ライセンスに含まれた

問題 9. gluck はいつ廃止になるか
□ A 6 月末
□ B Squeeze リリース時
□ C Lenny リリース時

問題 10. 新しく Debian keyring-mainter になったのは？
□ A Kenshi Muto
□ B Gunnar Wolf
□ C Jonathan McDowell

問題 11. 次期リリース: Squeeze でサポート対象から落ちそうなアーキテクチャは？
□ A alpha と hppa
□ B i386 と ia64
□ C s390 と sparc

問題 12. 次期リリース: Squeeze で採用される Linux カーネルのバージョンは？
□ A 2.6.31
□ B 2.6.32
□ C 2.6.33

問題 13. 追加されたパッケージのオートリジェクトは何がトリガーになる？
□ A 妻帯者か否か
□ B FTBFS のチェック
□ C lintian によるパッケージチェック

問題 14. Debconf10 はいつから開催されることになったか
□ A 2010 年 8 月 1 日
□ B 2010 年 1 月 1 日
□ C 2010 年 4 月 1 日

問題 15. Eugene V. Lyubimkin が発表した APT の代替アプリケーションは？
□ A debmoe
□ B cupt
□ C saitama

18 Debian Trivia Quiz 問題回答

上川 純一

1. A
2. A
3. A
4. A
5. B
6. B
7. B
8. A
9. A
10. B
11. A
12. B
13. C
14. A
15. B

19 索引

『あんどきゅめんてっど でびあん』について

本書は、東京および関西周辺で毎月行なわれている『東京エリア Debian 勉強会』および『関西エリア Debian 勉強会』で使用された資料・小ネタ・必殺技などを一冊にまとめたものです。収録範囲は東京エリアは 2009 年 6 月勉強会 (第 53 回) から 2009 年 11 月勉強会 (第 58 回)、関西エリアは第 24 回から第 29 回 まで。内容は無保証、つっこみなどがあれば勉強会にて。

あんどきゅめんてっど でびあん 2009 年冬号

2009 年 12 月 30 日　　初版第 1 刷発行
東京エリア Debian 勉強会/関西エリア Debian 勉強会　（編集・印刷・発行）